Mixed Signal Circuit Design with SystemC-AMS

Zainalabedin Navabi
ECE, University of Tehran

Slides prepared by Katayoon Basharkhaah, Ph.D. Student, ECE, University of Tehran

Outline

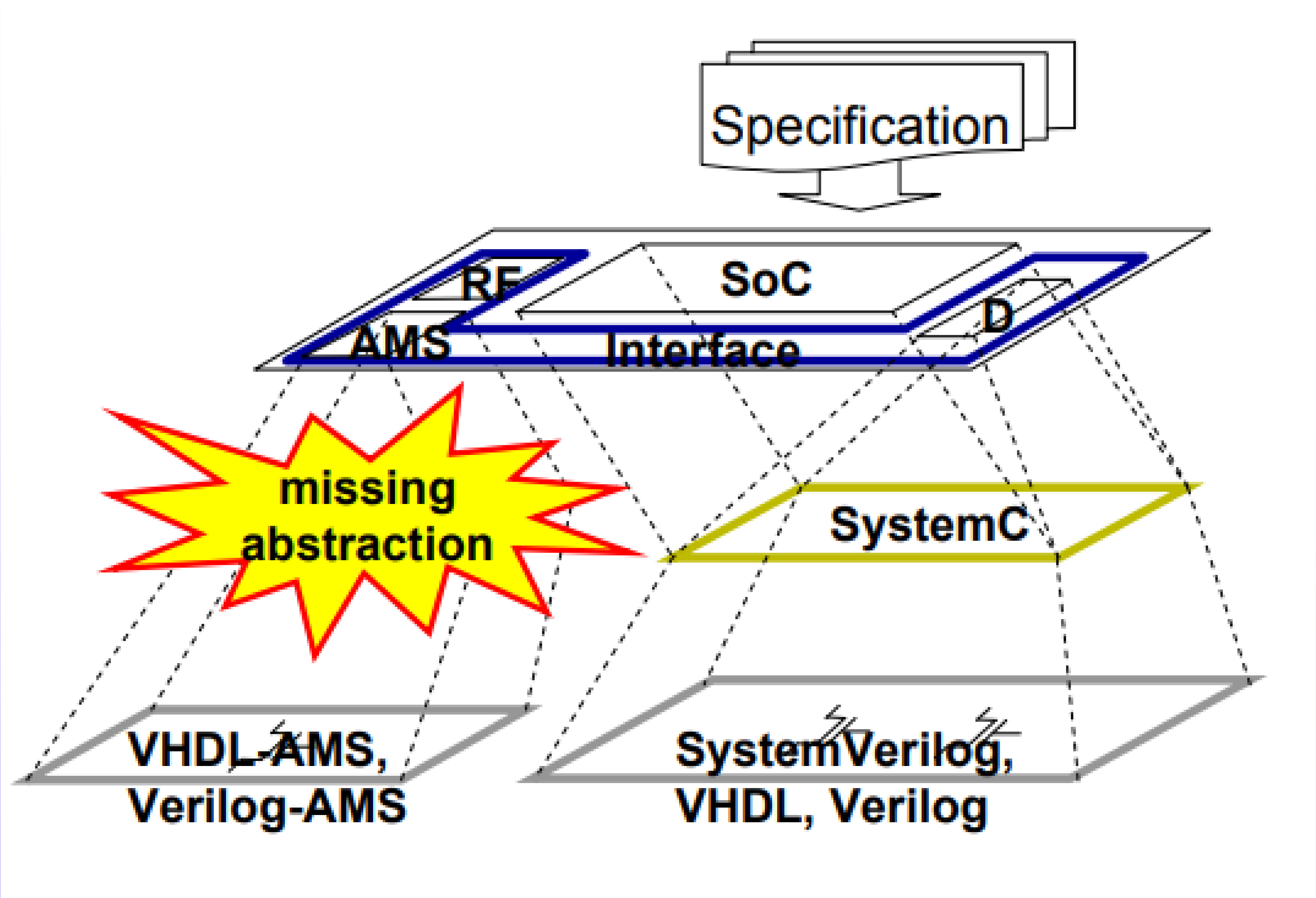
- Introduction
  - Applications
- Models of computation
  - ELN
  - LSF
  - TDF

Why AMS extension for SystemC?

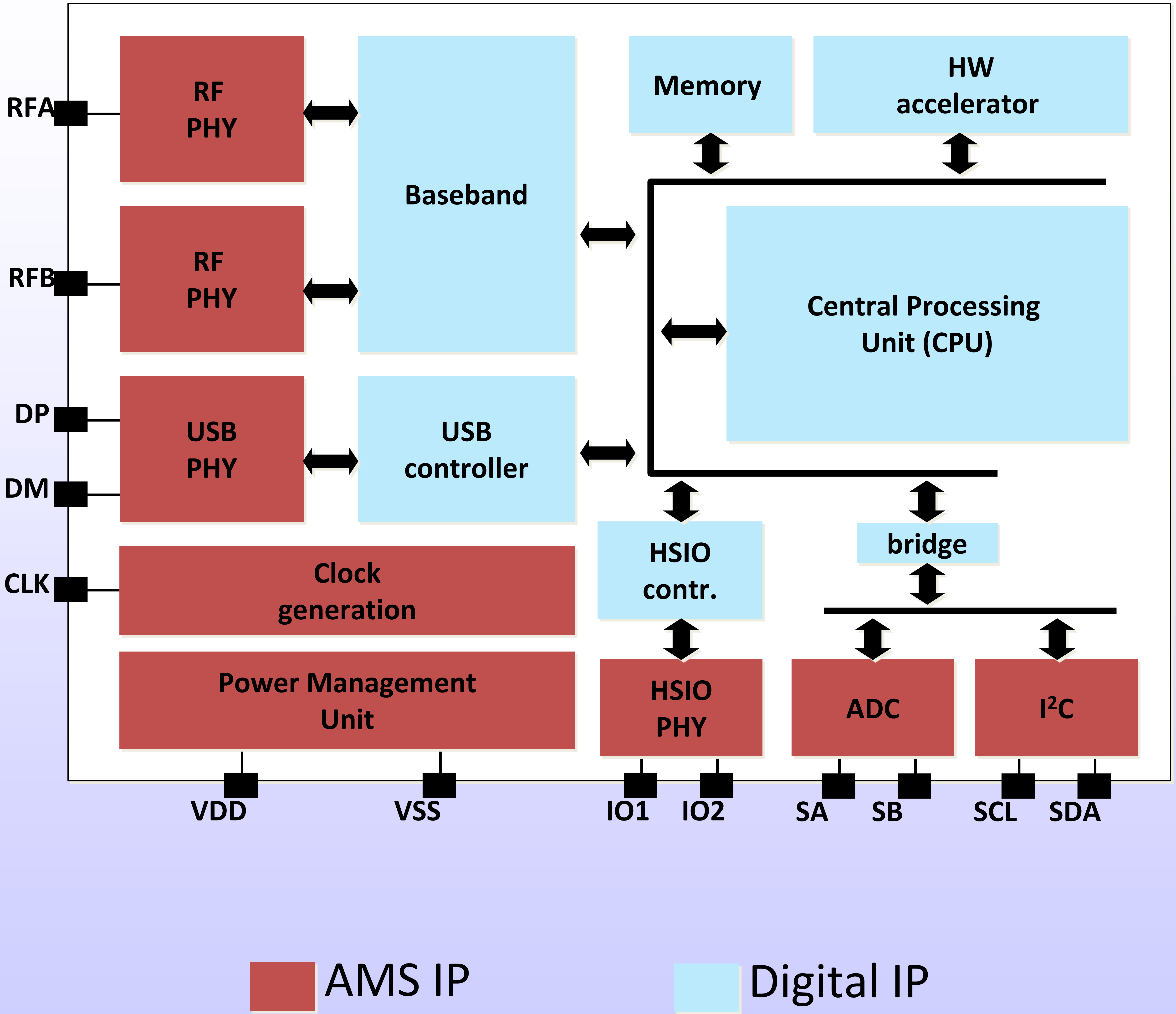
Functional

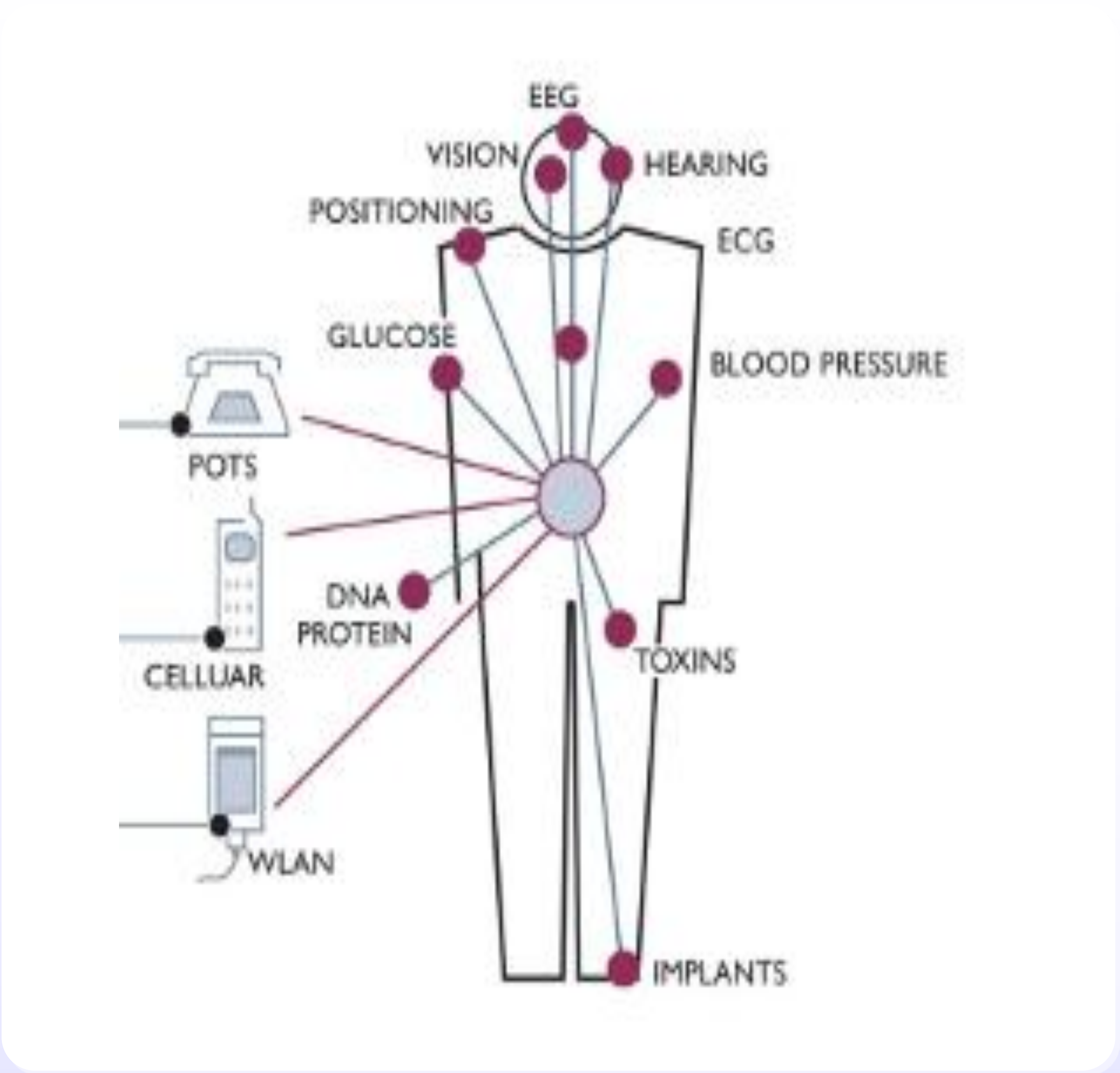
Architecture

Implementation




PHONAK


RFA
RFB
DP
DM
CLK
RF PHY
RF PHY
USB PHY
Baseband
USB controller
Memory
HW accelerator
Central Processing Unit (CPU)
Clock generation
Power Management Unit
HSIO contr.
bridge
HSIO PHY
ADC
I²C
VDD
VSS
IO1
IO2
SA
SB
SCL
SDA
AMS IP
Digital IP

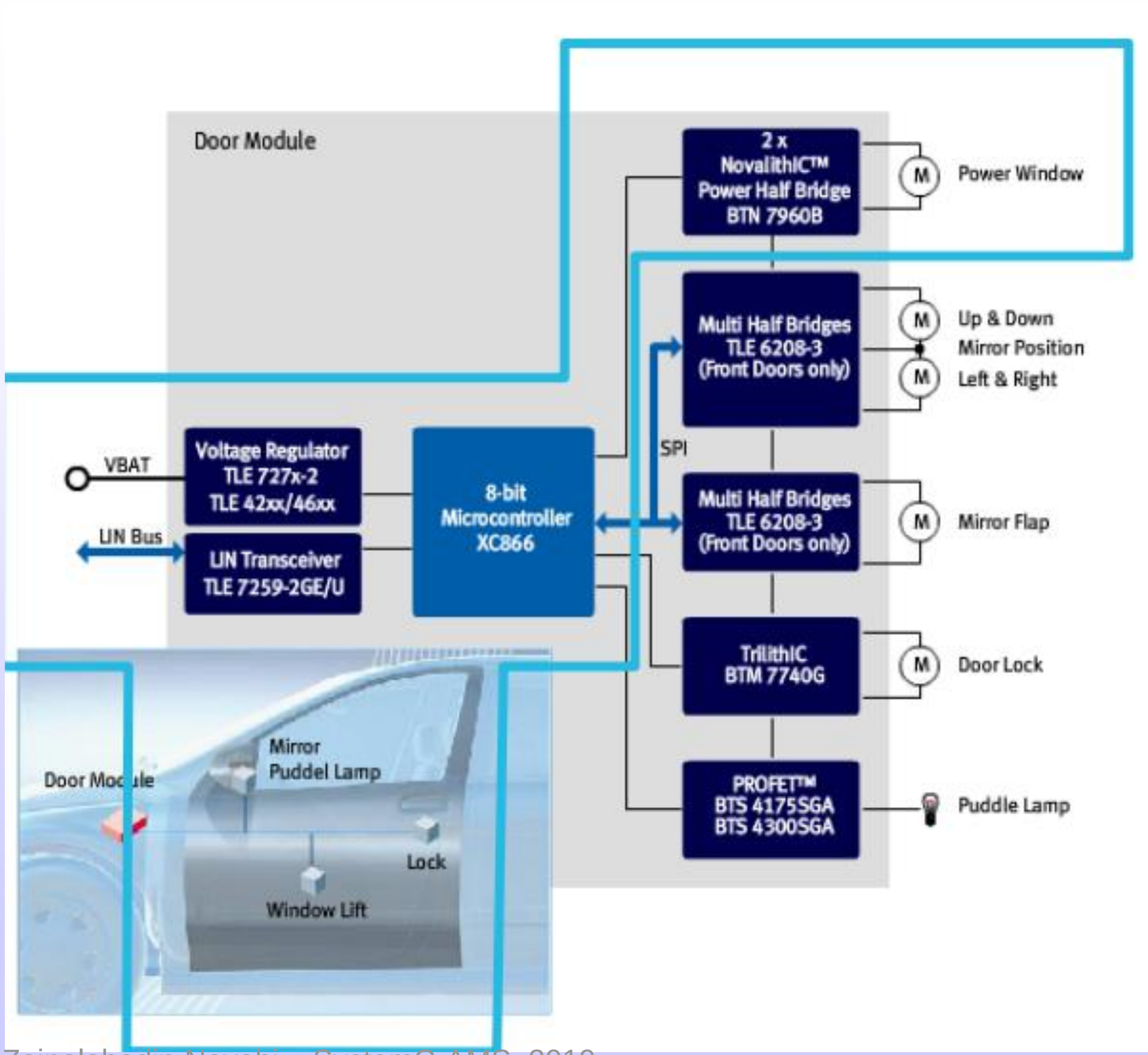

EEG
VISION
HEARING
POSITIONING
ECG
GLUCOSE
BLOOD PRESSURE
POTS
DNA PROTEIN
CELLUAR
TOXINS
WLAN
IMPLANTS


P5CD081 DEMO CARD


SPECIMEN
ELECTRONIC PASSPORT

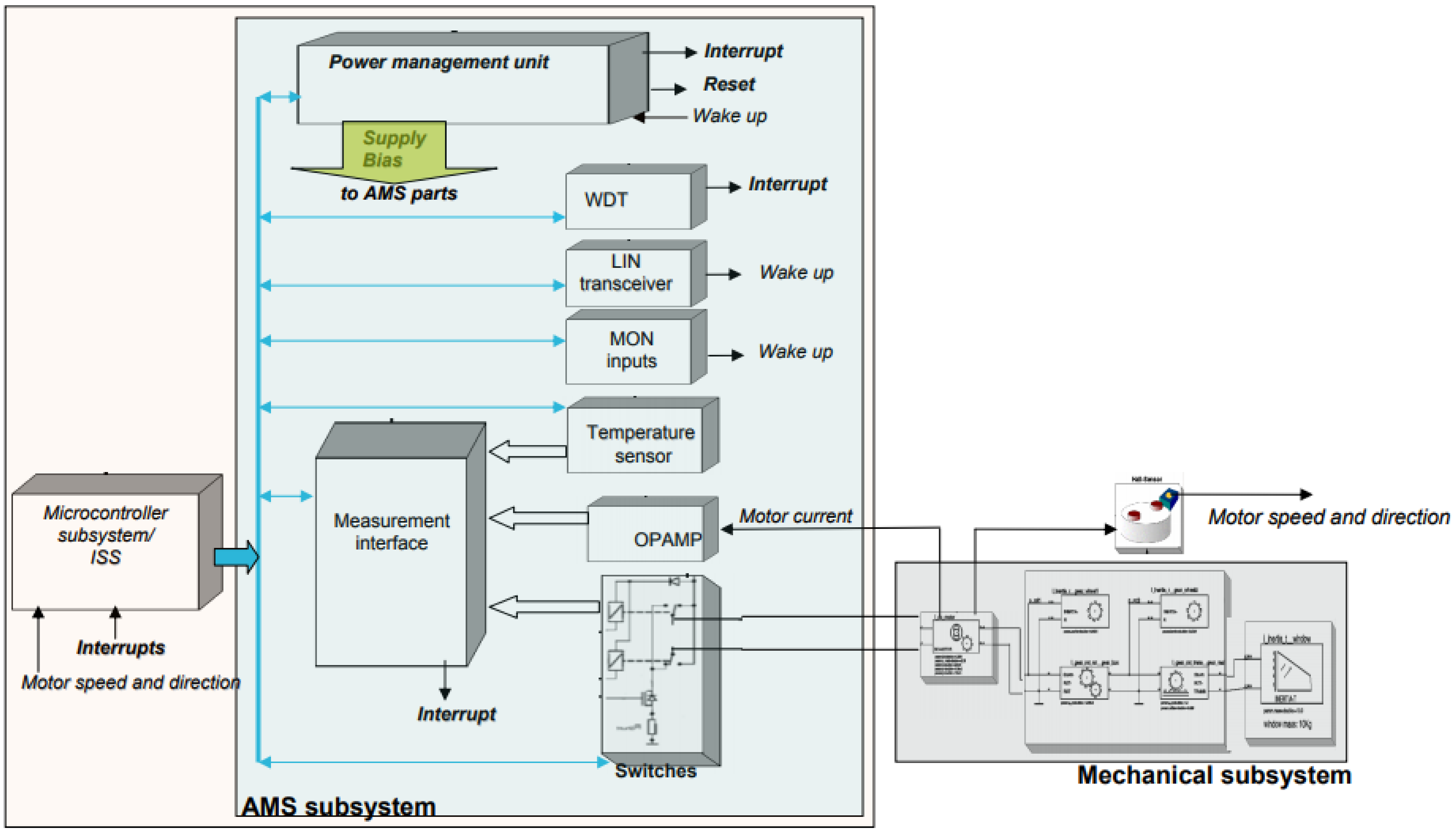
Applications

- Window lifter
  subset of door ECU



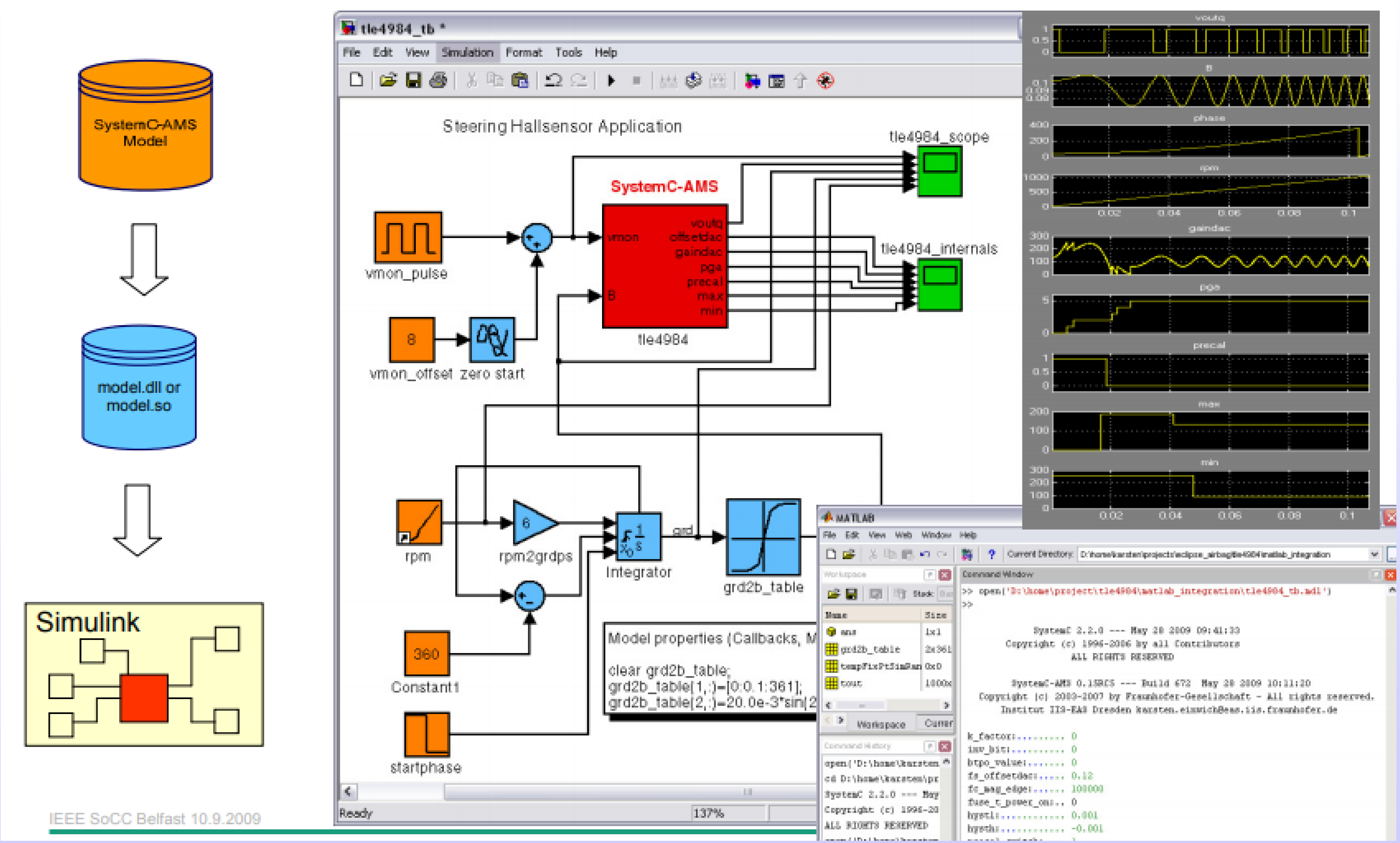
Applications

- Model overview



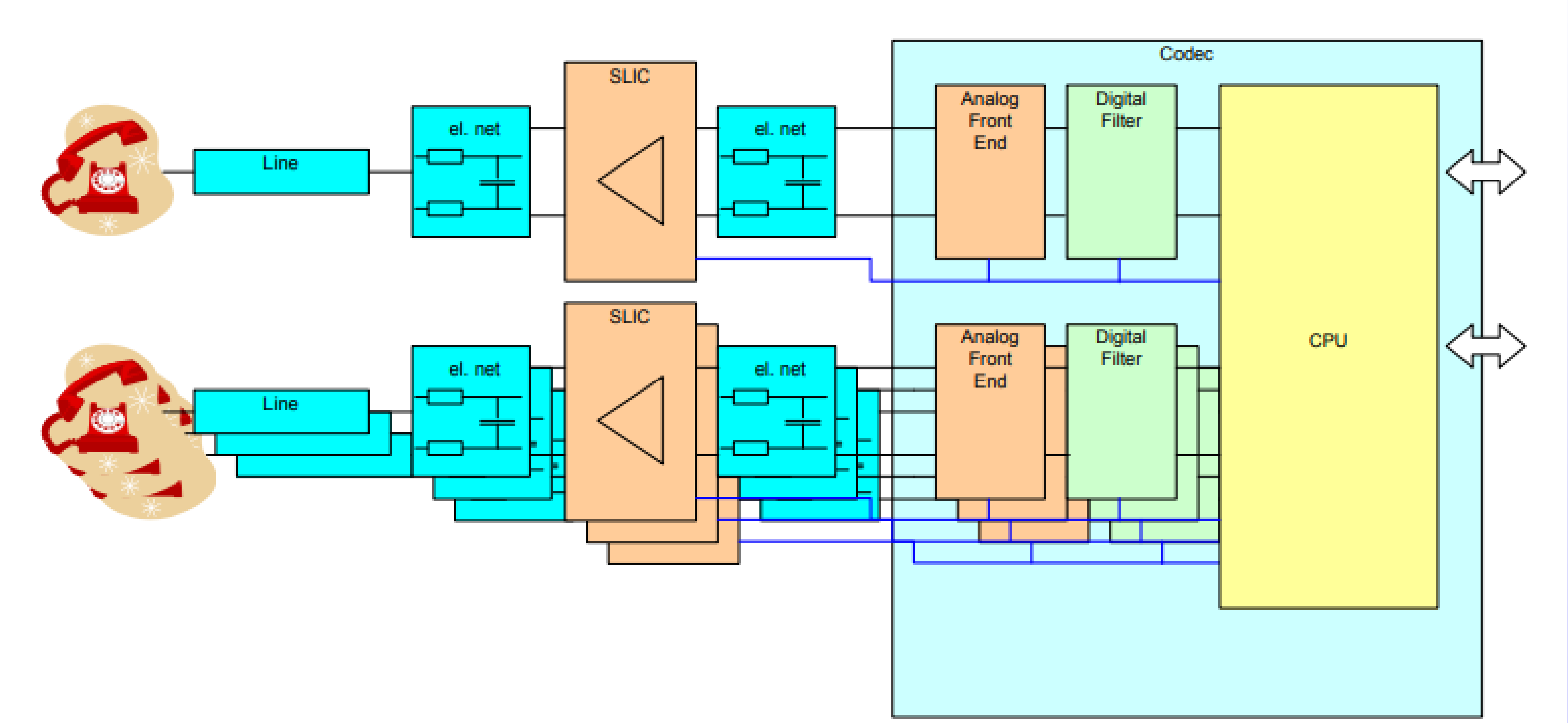
Applications

- Model exchange via simulink



Applications

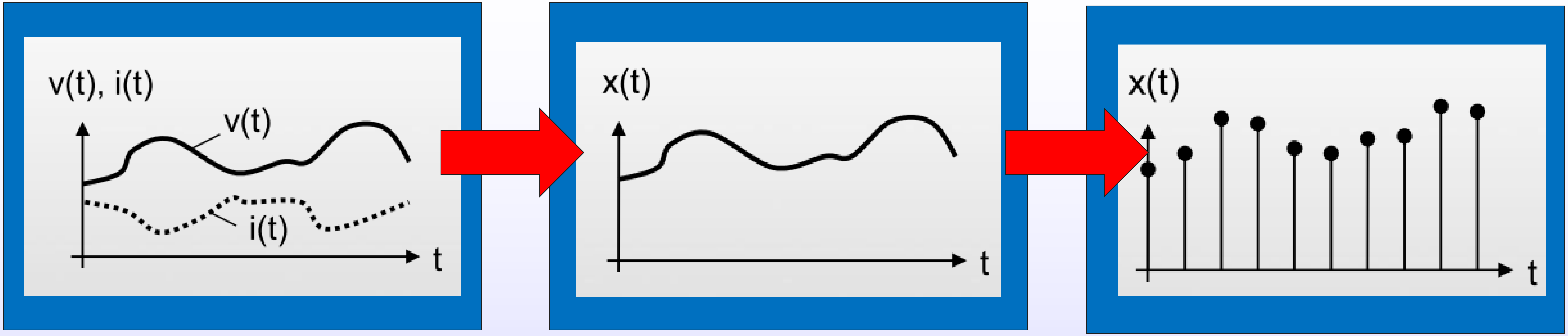
- Plain Old Telephone Service system design



Model of Computation

- Electrical Linear Networks (ELN)
- Linear Signal Flow (LSF)
- Timed Data Flow (TDF)

Abstraction of analog signals



Electrical Linear Network

- Conservative description represented by two dependent quantities, being the voltage v(t) and the current i(t)
- Continuous in time and value
- Analog (linear) solver will resolve the Kirchhoff's Law

Linear Signal Flow

- Non-conservative description represented by single quantity x(t), to represent e.g. the voltage or current (not both)
- Continuous in time and value

Timed Data Flow

- Non-conservative description represented by single quantity x(t).
- Discrete ti… only… arbit…

Perfect model abstraction

ELN Modeling Primitive Modules

- Sources (voltage or current)
- Linear lumped elements (resistors, capacitors, inductors)
- Linear distributed elements (transmission lines)
- Ideal amplifier
- Ideal transformer
- Linear gyrator
- Ideal switches

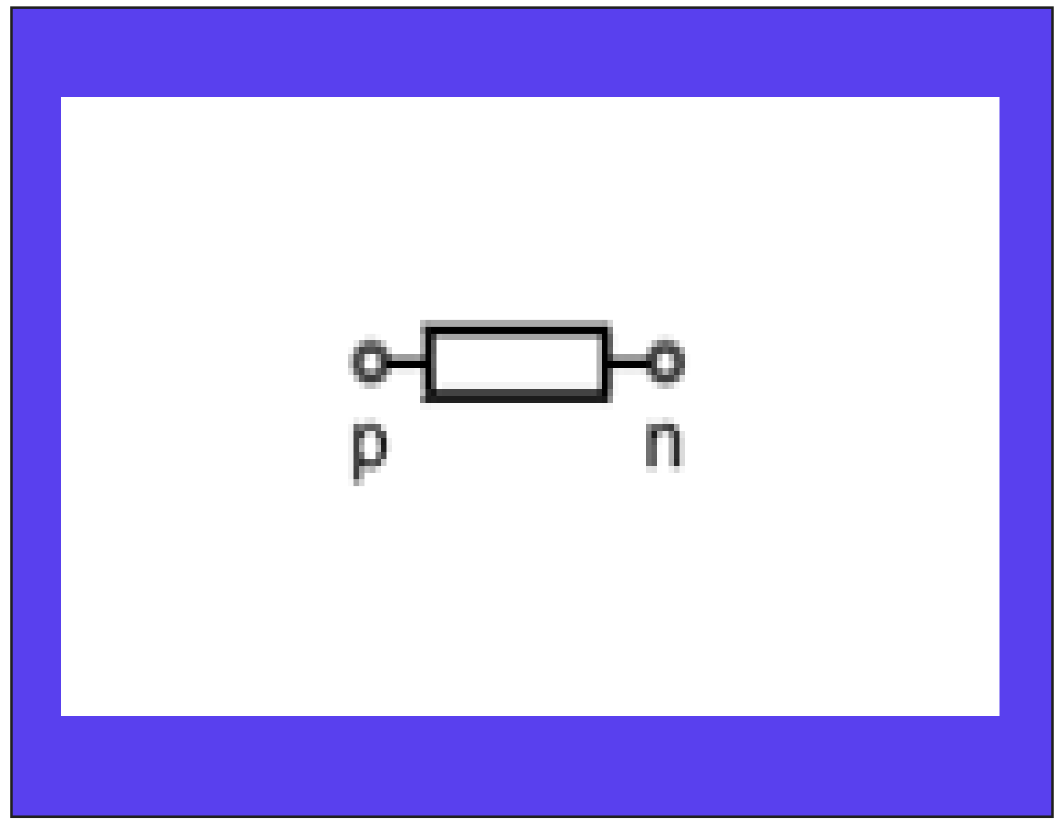
ELN Modeling Primitive Modules

- Resistor

Definition

Sca_eln::sca_r (nm, value)

Symbol



Equation

$$v_{p,n}(t) = i_{p,n}(t) \cdot value$$

Parameters

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
value	double	1.0	Resistance in Ohm

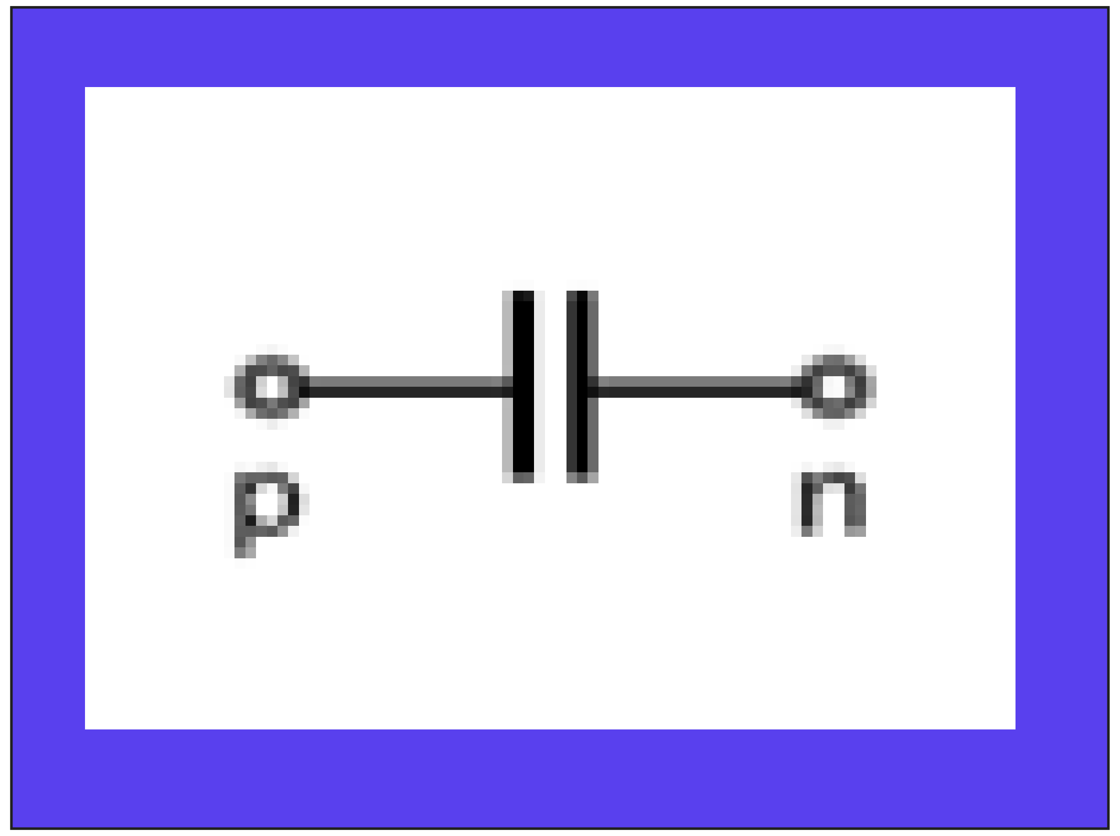
ELN Modeling Primitive Modules

- Capacitor

Definition

Sca_eln::sca_c (nm, value, q0)

Symbol



Equation

$$i_{p,n}(t) = \frac{d(value \cdot v_{p,n}(t) + q_0)}{dt}$$

Parameters

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
value	double	1.0	Capacitance in Farad
q0	double	0.0	Initial charge in Coulomb

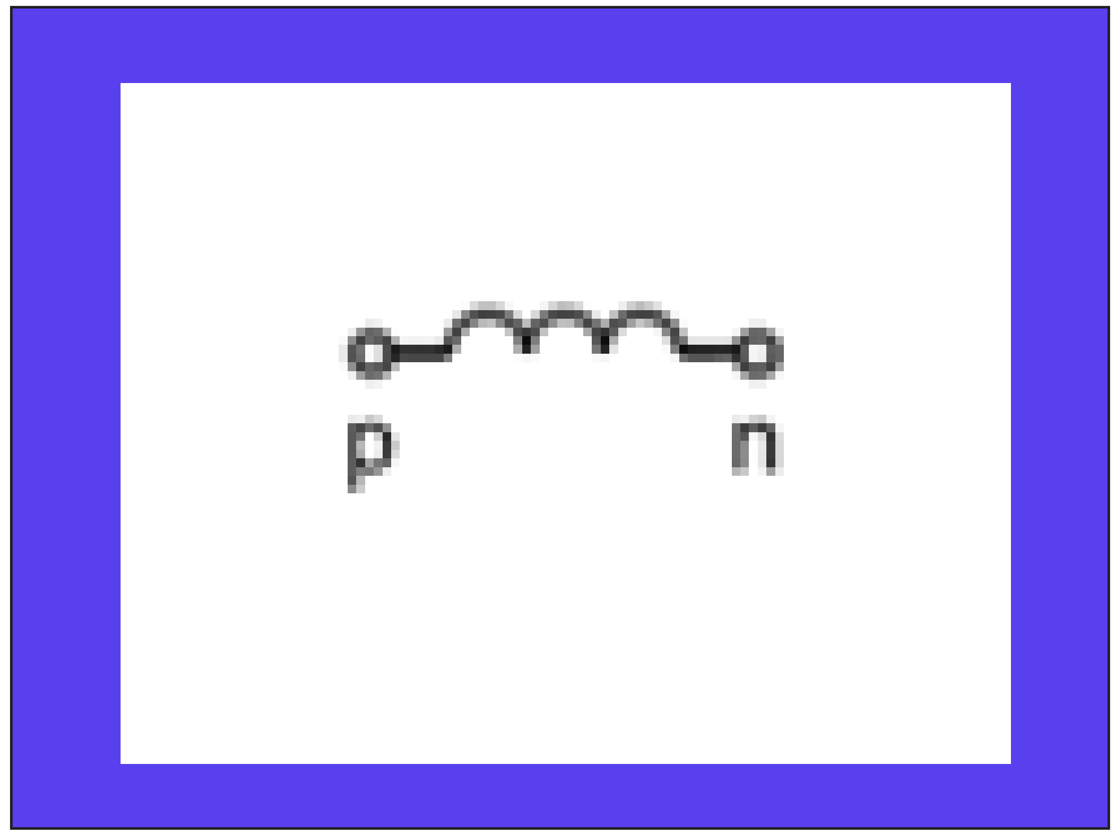
ELN Modeling Primitive Modules

Inductor

Definition

Sca_eln::sca_l (nm, value, phi0)

Symbol



Equation

$$v_{p,n}(t) = \frac{d\left(value \cdot i_{p,n}(t) + phi_0\right)}{dt}$$

Parameters

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
value	double	1.0	Inductance in Henry
phi0	double	0.0	Initial magnetic flux in Weber

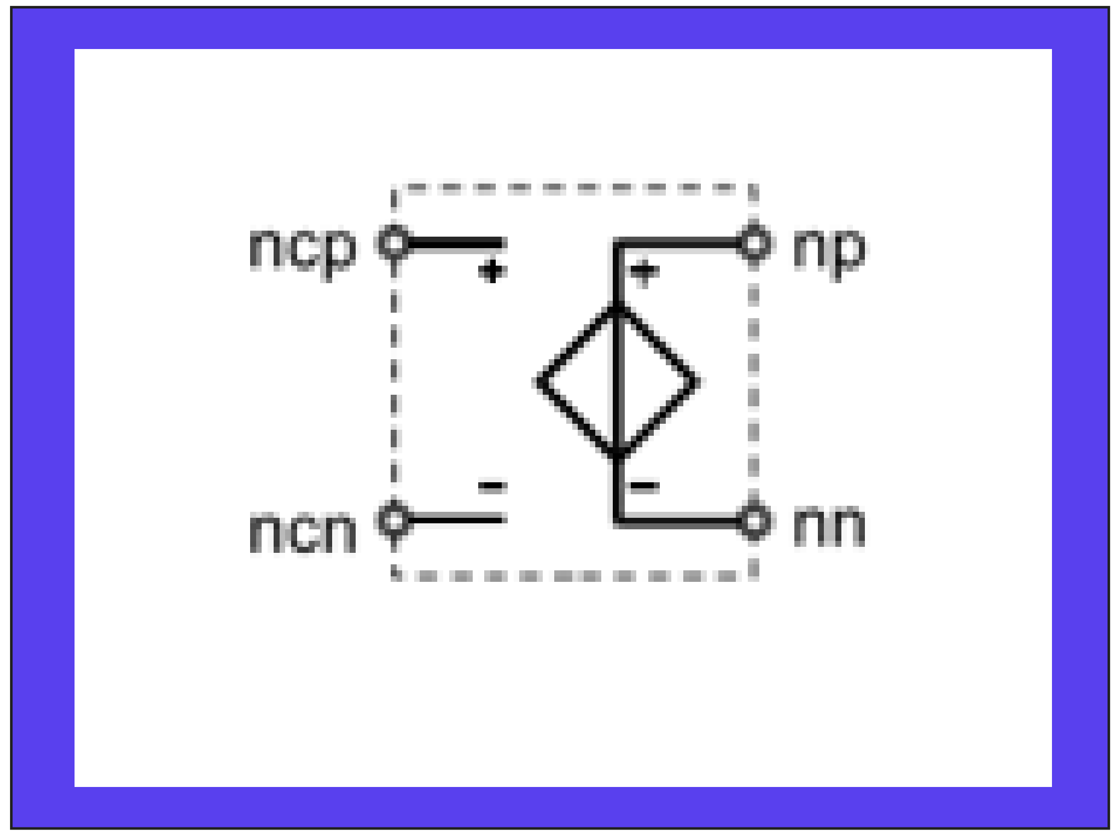
ELN Modeling Primitive Modules

- Voltage controlled voltage source

Definition

Sca_eln::sca_vcvs (nm, value)

Symbol



Equation

$$v_{np,nn}(t) = value \cdot v_{ncp,ncn}(t)$$

Parameters

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
value	double	1.0	Scale coefficient of the control voltage

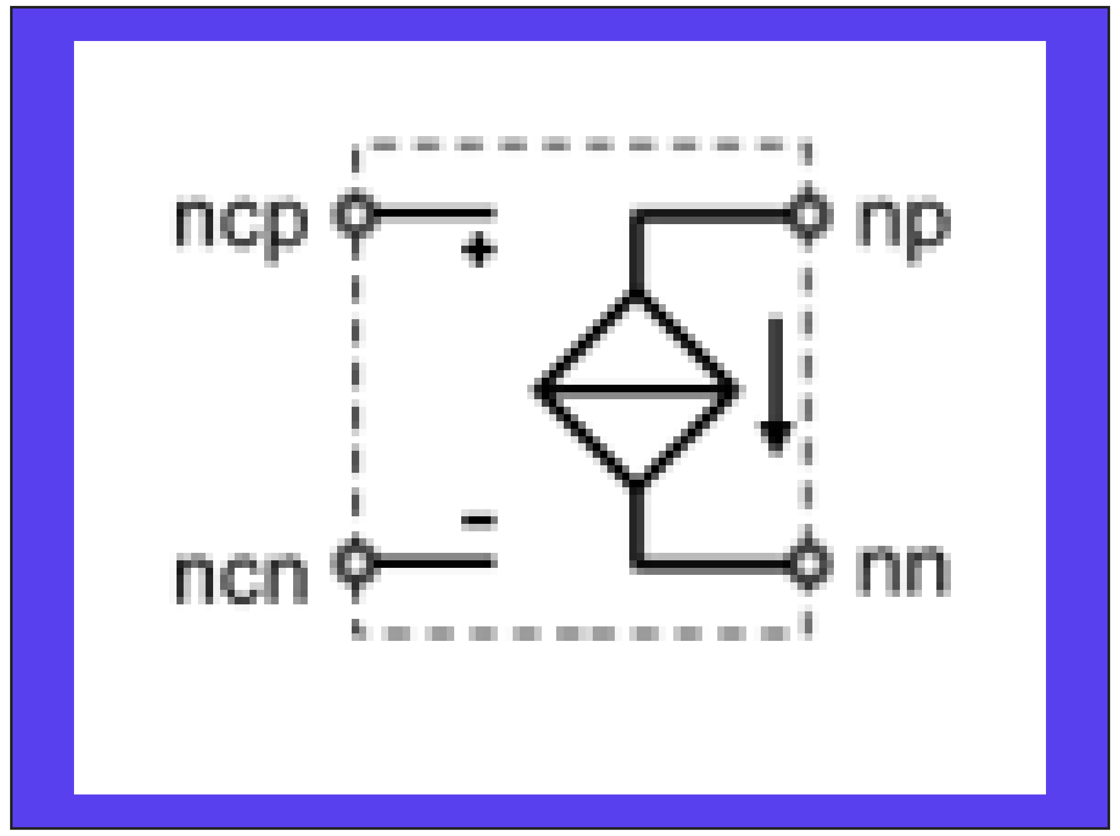
ELN Modeling Primitive Modules

- Voltage controlled current source

Definition

Sca_eln::sca_vccs (nm, value)

Symbol



Equation

$$i_{np,nn}(t) = value \cdot v_{ncp,ncn}(t)$$

Parameters

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
value	double	1.0	Scale coefficient in Siemens of the control voltage

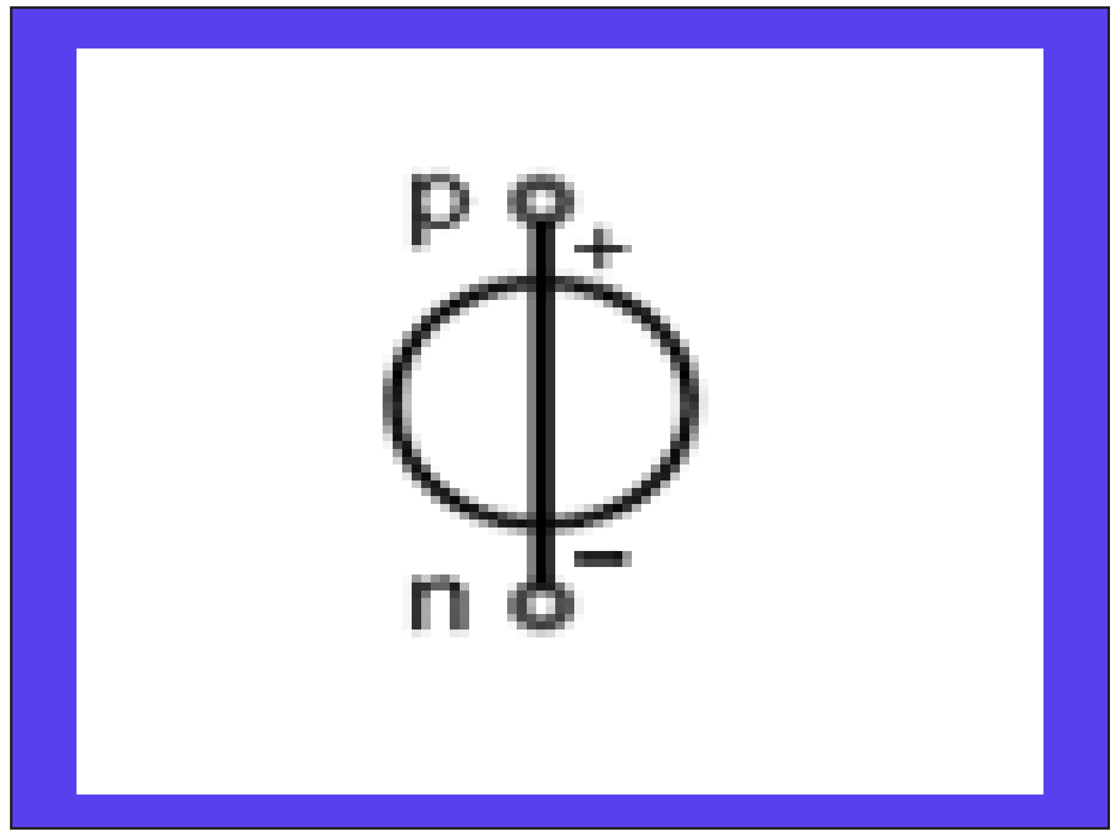
ELN Modeling Primitive Modules

- Independent Voltage source

Definition

Sca_eln::sca_vsource (nm, init_value, offset, amplitude, frequency, phase, delay, ac_amplitude, ac_phase, ac_noise)_amplitude

Symbol



Equation

For time-domain simulation:

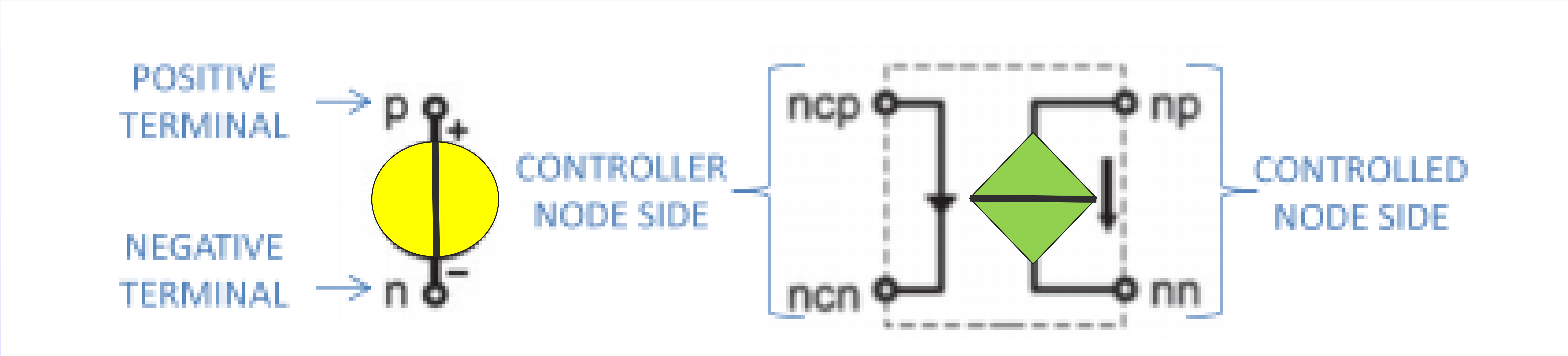
$$v_{p,n}(t) = \begin{cases} init\_value & t < delay \\ offset + amplitude \cdot \sin(2\pi \cdot frequency \cdot (t - delay) + phase) & t \geq delay \end{cases}$$

For small-signal frequency-domain simulation:

$$v_{p,n}(f) = ac\_amplitude \cdot \{\cos(ac\_phase) + j \cdot \sin(ac\_phase)\}$$

For small-signal frequency-domain noise simulation:

ELN Modeling Primitive Modules



ELN Modeling Primitive Modules

$V(a) = 4.02\ V(b) - 3.72\ I(c) + 8.01$

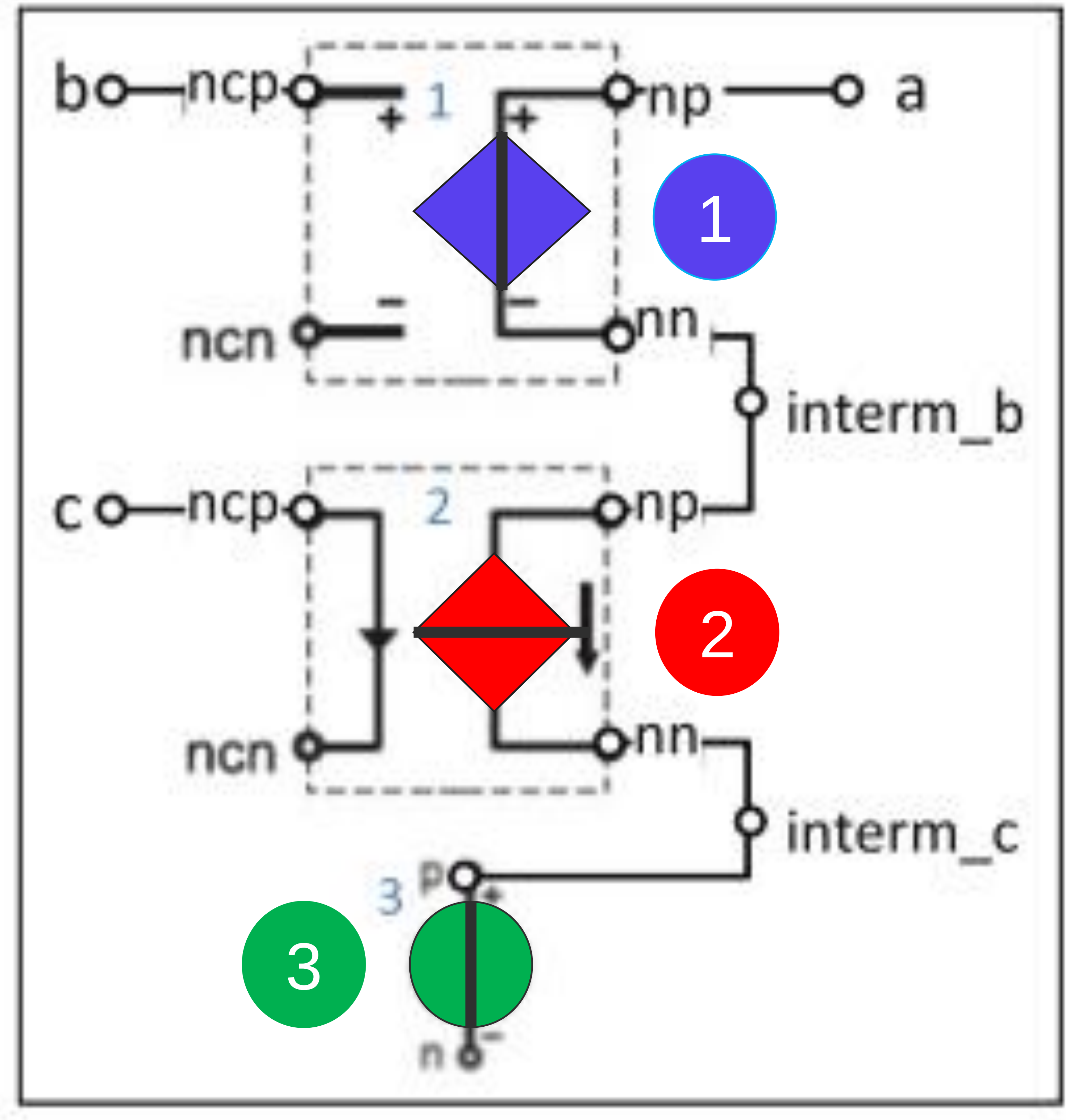
1 2 3

```
vcvs_b= sca_vcvs (bb, +4.02);
vcvs_b -> np(a);
vcvs_b -> nn(interm_b);
vcvs_b -> ncp(b);
vcvs_b -> ncn(gnd);
```

1

```
ccvs_c= sca_ccvs (cc, -3.72);
ccvs_c -> np(interm_b);
ccvs_c -> nn(interm_c);
ccvs_c -> ncp(c);
ccvs_c -> ncn(gnd);
```

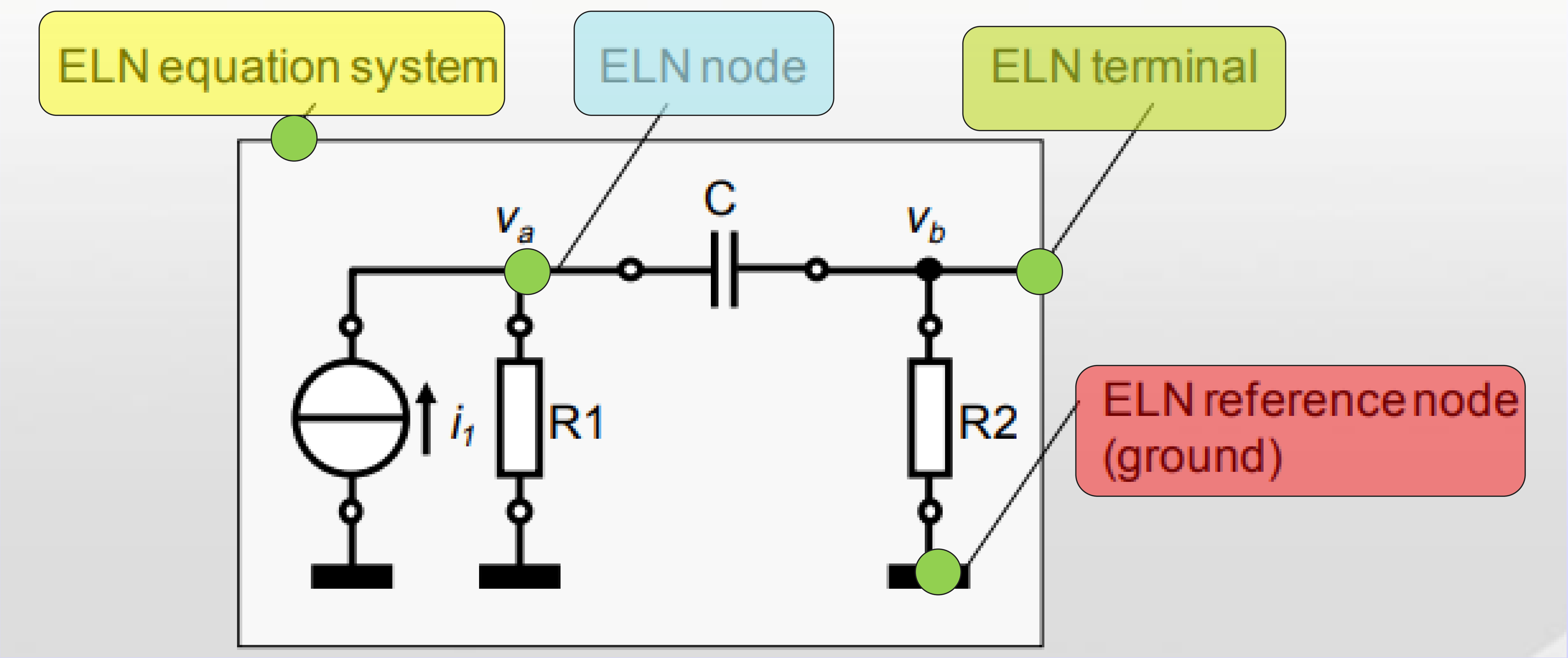
2




```
vcs= sca_vsource (vs, +8.01);
vcs -> np(interm_c);
vcvs_b -> nn(gnd);
```

3

ELN Basics



ELN equation system
ELN node
ELN terminal
ELN reference node (ground)
C
v_a
v_b
i_1
R1
R2

ELN Terminals

- Objects that can be used to connect several ELN models, using ELN nodes bound to this terminal
- They can not be defined as input or output ports
- They do not provide read or write access methods

```
SC_MODULE(my_eln_model)
{
    // terminal declarations
    sca_eln::sca_terminal p;
    sca_eln::sca_terminal n;

    SC_CTOR(my_eln_model) : p("p"), n("n")
    {
        // model implementation here
    }
```

```
// using the constructor initialization-list to assign the names to the declared ELN nodes
SC_CTOR(my_eln_module) : net1("net1"), gnd("gnd") {}
```

ELN Nodes

- Used to connect ELN primitive modules sharing the same node (also called *net*)

- Two classes of ELN nodes:

  - ELN node of class **sca_eln::sca_node**
  - ELN reference node (ground) of class **sca_eln::sca_node_ref**

```
// node declarations
sca_eln::sca_node net1; // ELN node (called "net1")
sca_eln::sca_node_ref gnd; // ELN reference node (called ground, "gnd")
```

```
// using the constructor initialization-list to assign the names to the declared ELN nodes
SC_CTOR(my_eln_module) : net1("net1"), gnd("gnd") {}
```

Structural Composition of ELN Modules

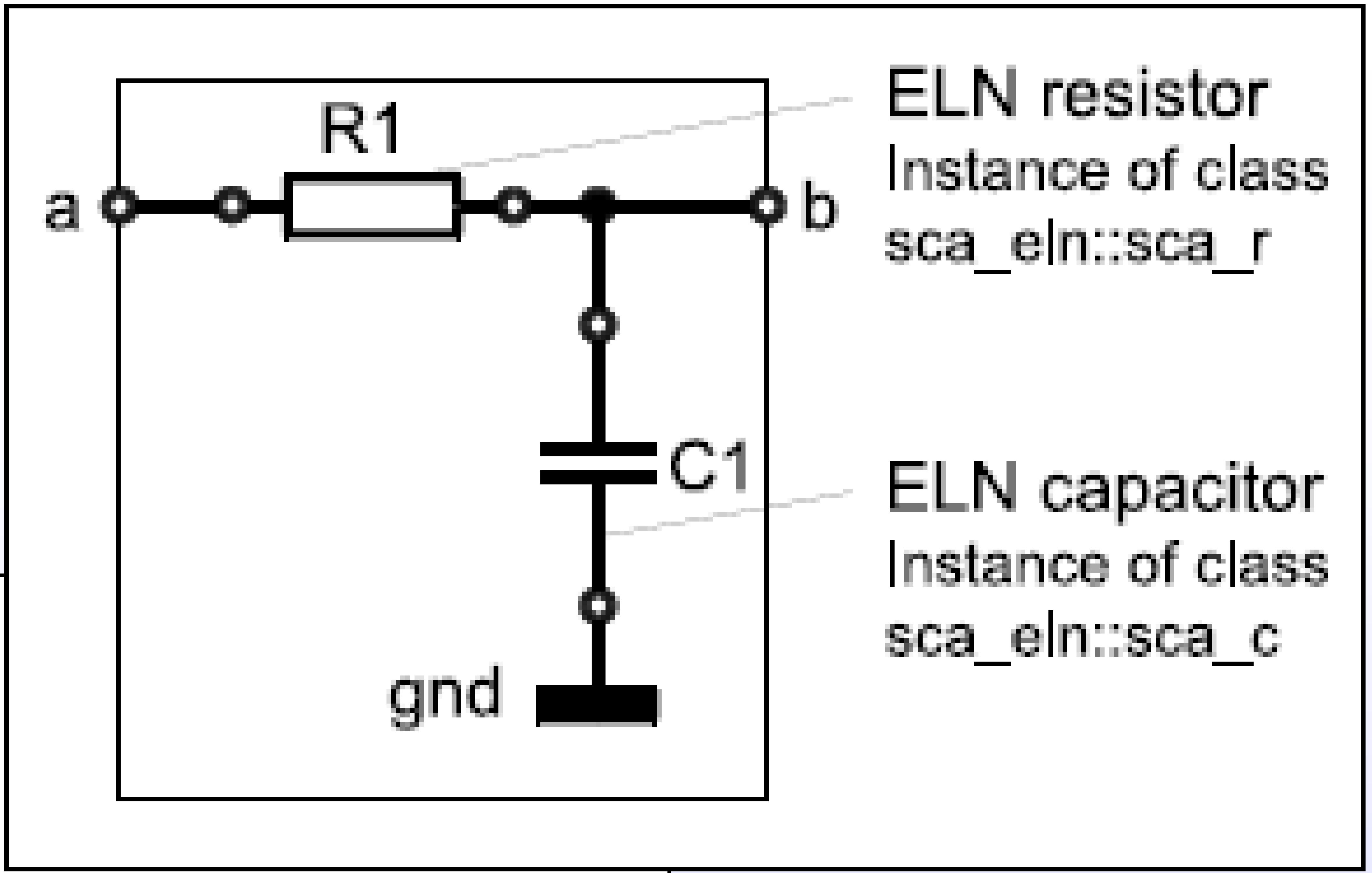
- ELN modules should be instantiated as child modules inside a regular SystemC parent module (SC_MODULE) or by deriving publicly from sc_core::sc_module.

- The parameterization of the instantiated modules and the interconnection of the modules is done in the constructor (e.g. SC_CTOR)

- Notes :
  - An ELN terminal should be bound to exactly one ELN node or reference node throughout the whole hierarchy.

Continuous Modeling

Example : A 1st order low pass filter




```
SC_MODULE(my_eln_filter)
{
    sca_eln::sca_terminal a;
    sca_eln::sca_terminal b;

    sca_eln::sca_r r1;
    sca_eln::sca_c c1;

    my_eln_filter( sc_core::sc_module_name, double r1_value, double c1_value )
    : a("a"), b("b"), r1("r1", r1_value), c1("c1", c1_value), gnd("gnd"),
    {
        r1.n(a);
        r1.p(b);
        c1.n(b);
        c1.p(gnd);
    }

    private:
        sca_eln::sca_node_ref gnd;
};
```


© Zainalabedin Navabi – SystemC AMS, 2019

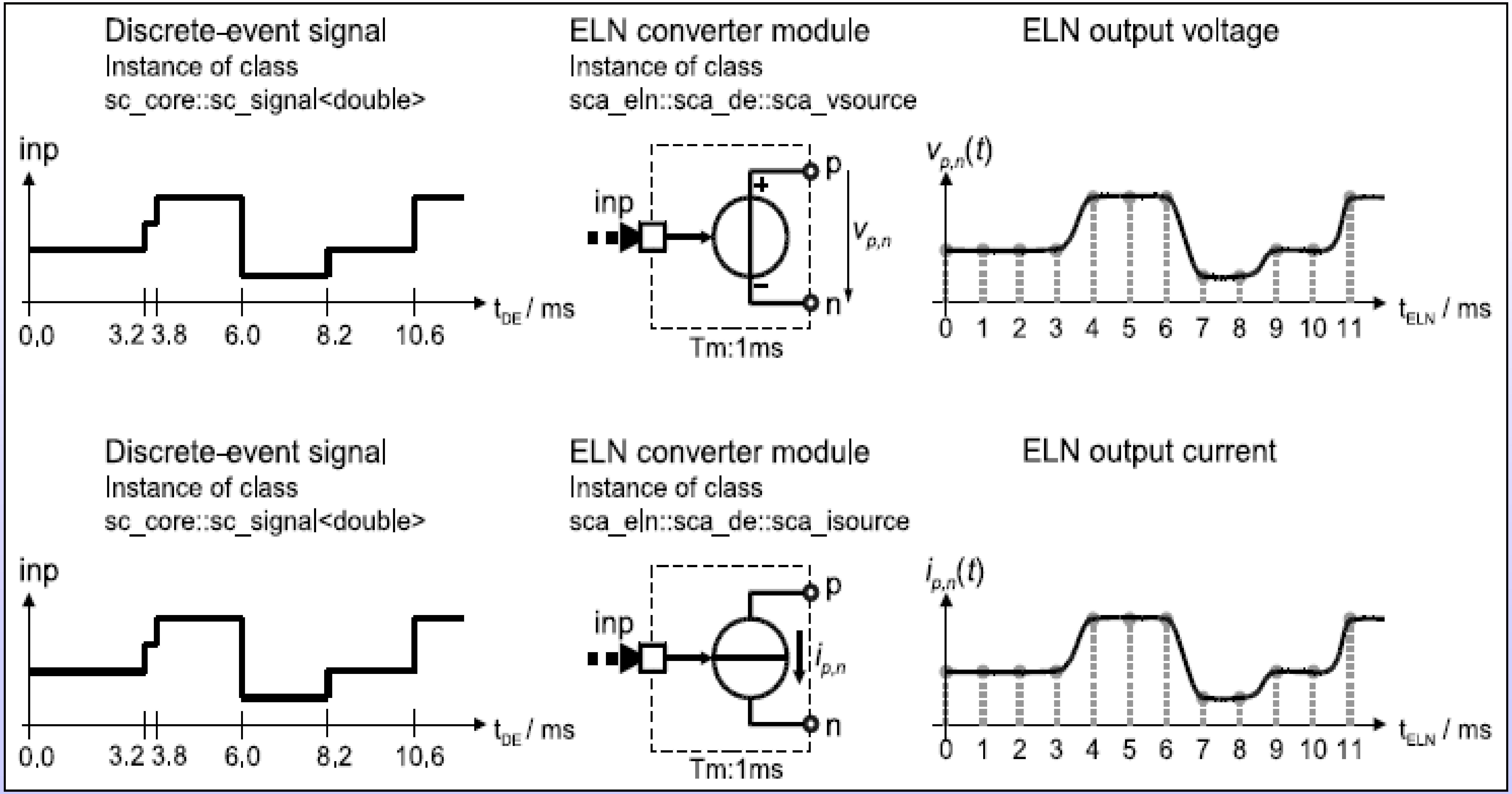
Interaction between ELN and Discrete-Event Model

- Specialized ELN primitive modules with ports to the discrete-event domain are available (called *converter modules*) to establish an interface to convert and transfer data from one model of computation to the other.

```
sca_eln::sca_de::sca_vsource
sca_eln::sca_de::sca_isource
```

- Read a discrete-event signal representing a real value and convert this value to an electrical voltage or current respectively

Reading from and Writing to Discrete-Event Models

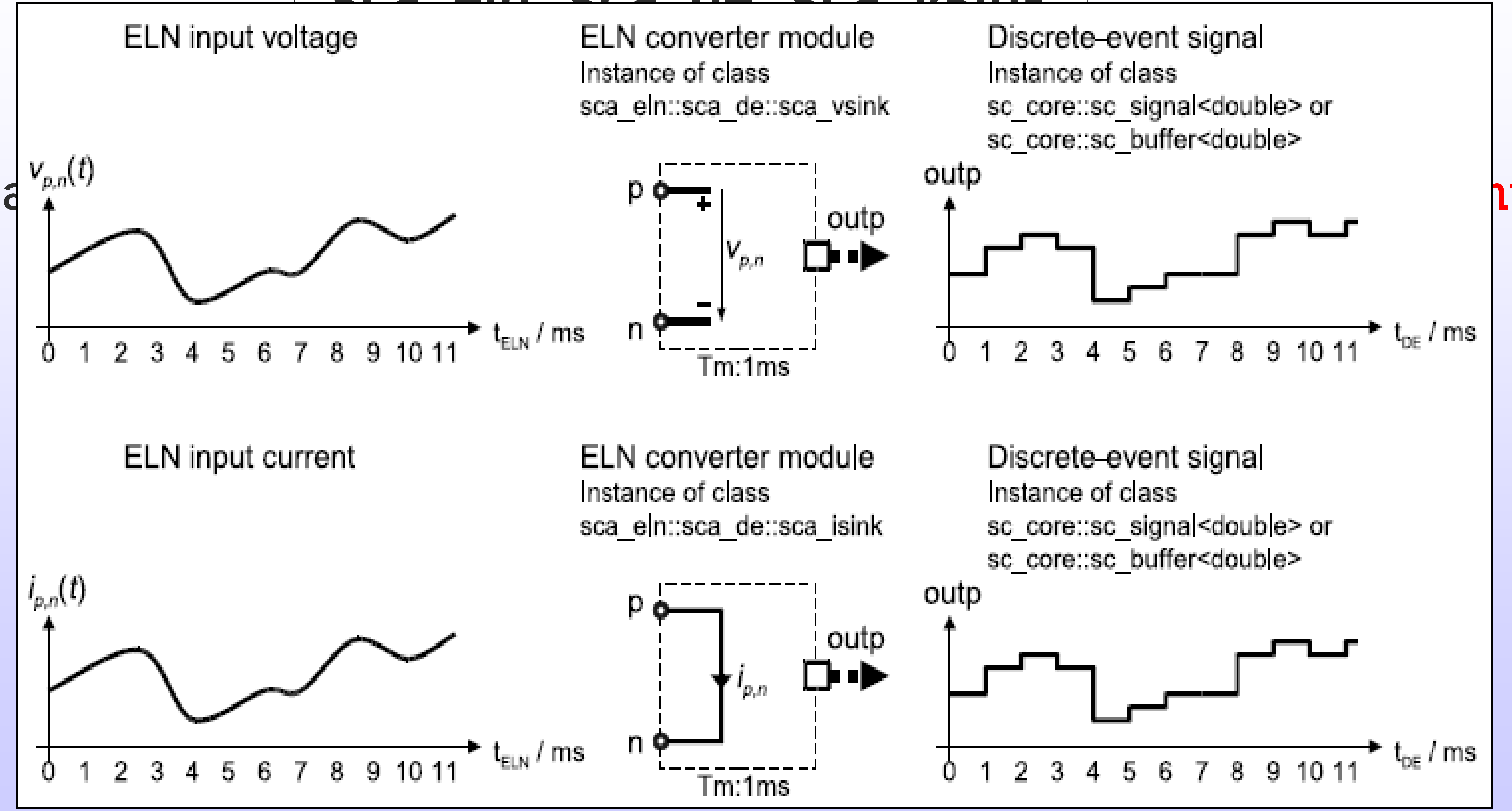


© Zainalabedin Navabi – SystemC AMS, 2019

Reading from and Writing to Discrete-Event Models

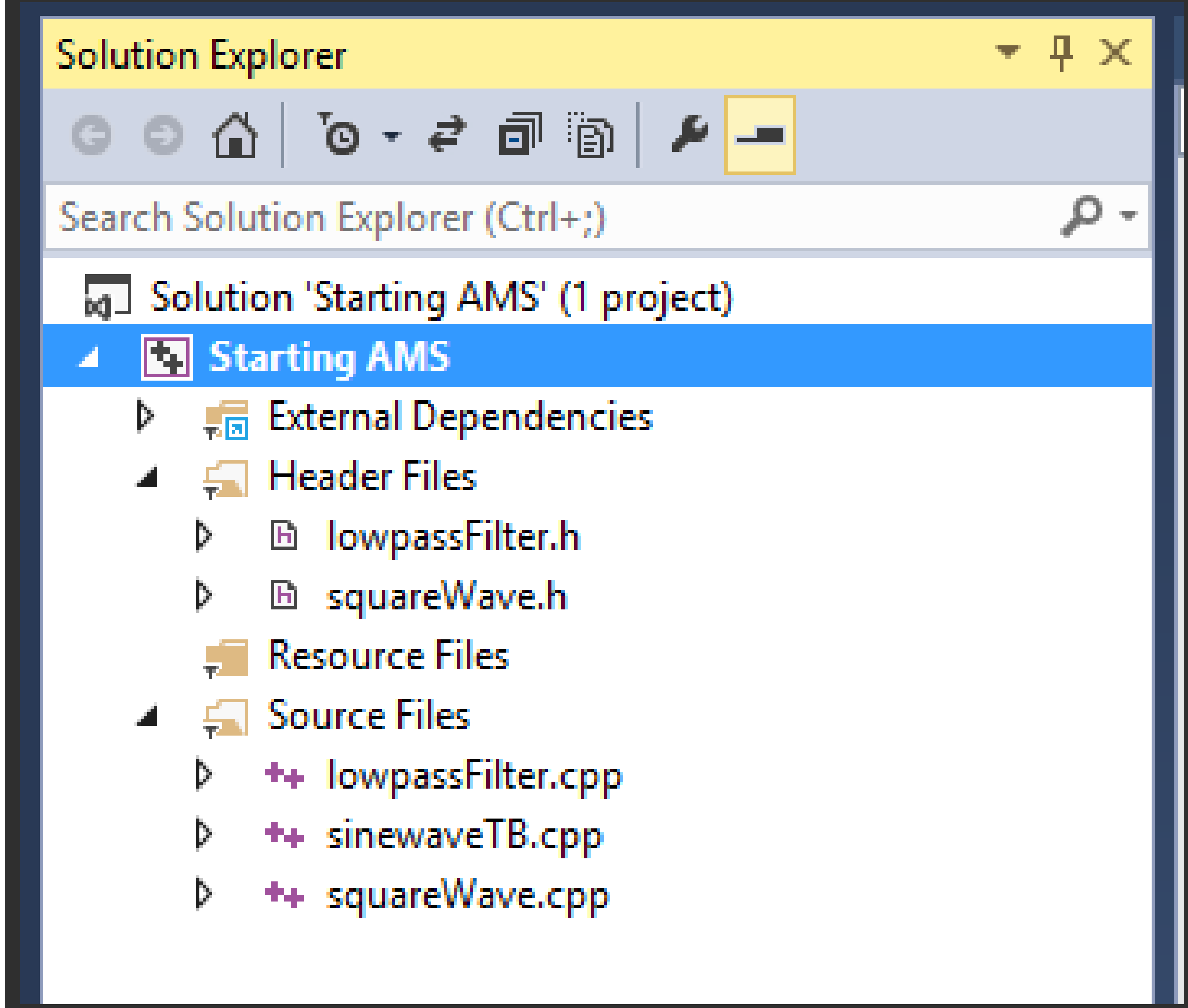
sca_eln::sca_de::sca_vsink

Convert a … nt signal



© Zainalabedin Navabi – SystemC AMS, 2019

Example - Filter



```cpp
#include <systemc.h>
#include <systemc-ams.h>

SC_MODULE(lowpassFilter){
    sc_in  <double> in;
    sc_out  <double> out;

    // primitive module instantiations
    sca_eln::sca_r r1;
    sca_eln::sca_c c1;
    sca_eln::sca_de_vsource vin;
    sca_eln::sca_de_vsink vout;

    SC_HAS_PROCESS(sc_module_name);
    lowpassFilter(sc_module_name, double r1_value, double c1_value);

private:
    sca_eln::sca_node a;
    sca_eln::sca_node b;
    sca_eln::sca_node_ref gnd;
};
```

squareWave.h | lowpassFilter.h | squareWave.cpp | sinewaveTB.cpp | lowpassFilter.cpp

Starting AMS | (Global Scope)

```
#include "lowpassFilter.h"

lowpassFilter::lowpassFilter (sc_module_name, double r1_value, double c1_value)
    :r1("r1", r1_value), c1("c1", c1_value), vin ("vin", 1.0), vout("vout", 1.0)
    {
        vin.p(a);
        vin.n(gnd);
        vin.inp(in);
        vin.set_timestep(1, SC_MS);

        r1.n(a);
        r1.p(b);

        c1.n(b);
        c1.p(gnd);

        vout.p(b);
        vout.n(gnd);
        vout.outp(out);
    }

```

squareWave.h | lowpassFilter.h | squareWave.cpp | sinewaveTB.cpp | lowpassFilter.cpp

Starting AMS | (Global Scope)

```cpp
#include <systemc-ams.h>

SC_MODULE(squareWave){
    sc_out <double> out;

    SC_CTOR(squareWave){
        SC_THREAD(wave);
    }
    void wave();
};
```

squareWave.h | lowpassFilter.h | squareWave.cpp | sinewaveTB.cpp | lowpassFilter.cpp

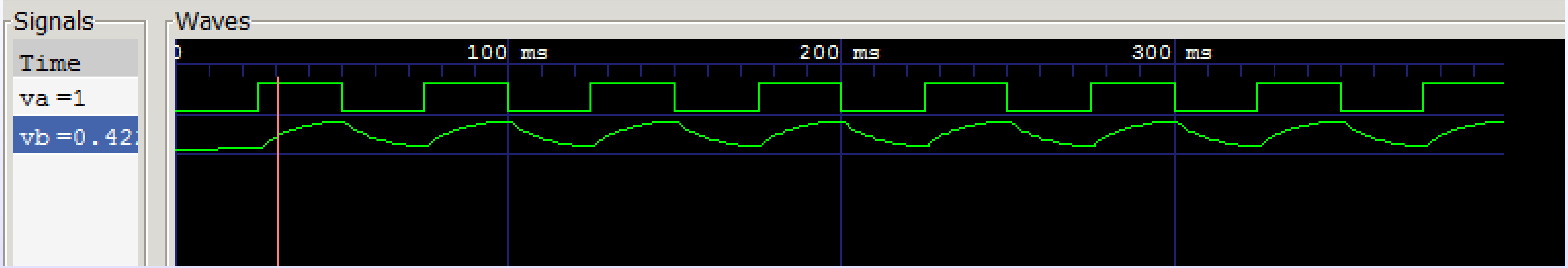
Starting AMS | (Global Scope)

```cpp
#include "squareWave.h"

void squareWave::wave()
{
    while (1){
        wait(25, SC_MS);
        out->write(1.0);
        wait(25, SC_MS);
        out->write(0.0);
    }
}
```

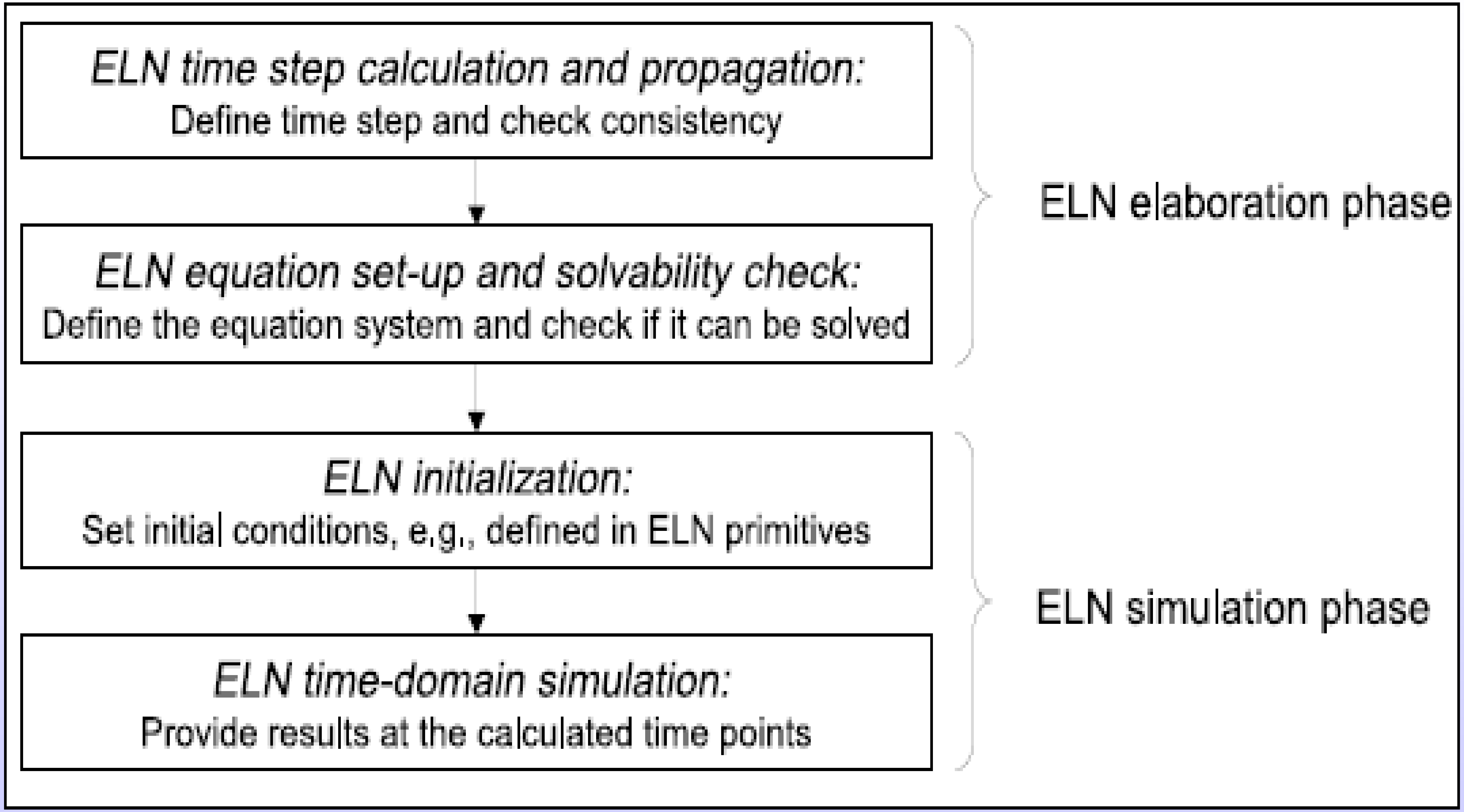
squareWave.h | lowpassFilter.h | squareWave.cpp | sinewaveTB.cpp | lowpassFilter.cpp

Starting AMS | (Global Scope) | sc_main(int argc, char * argv[])

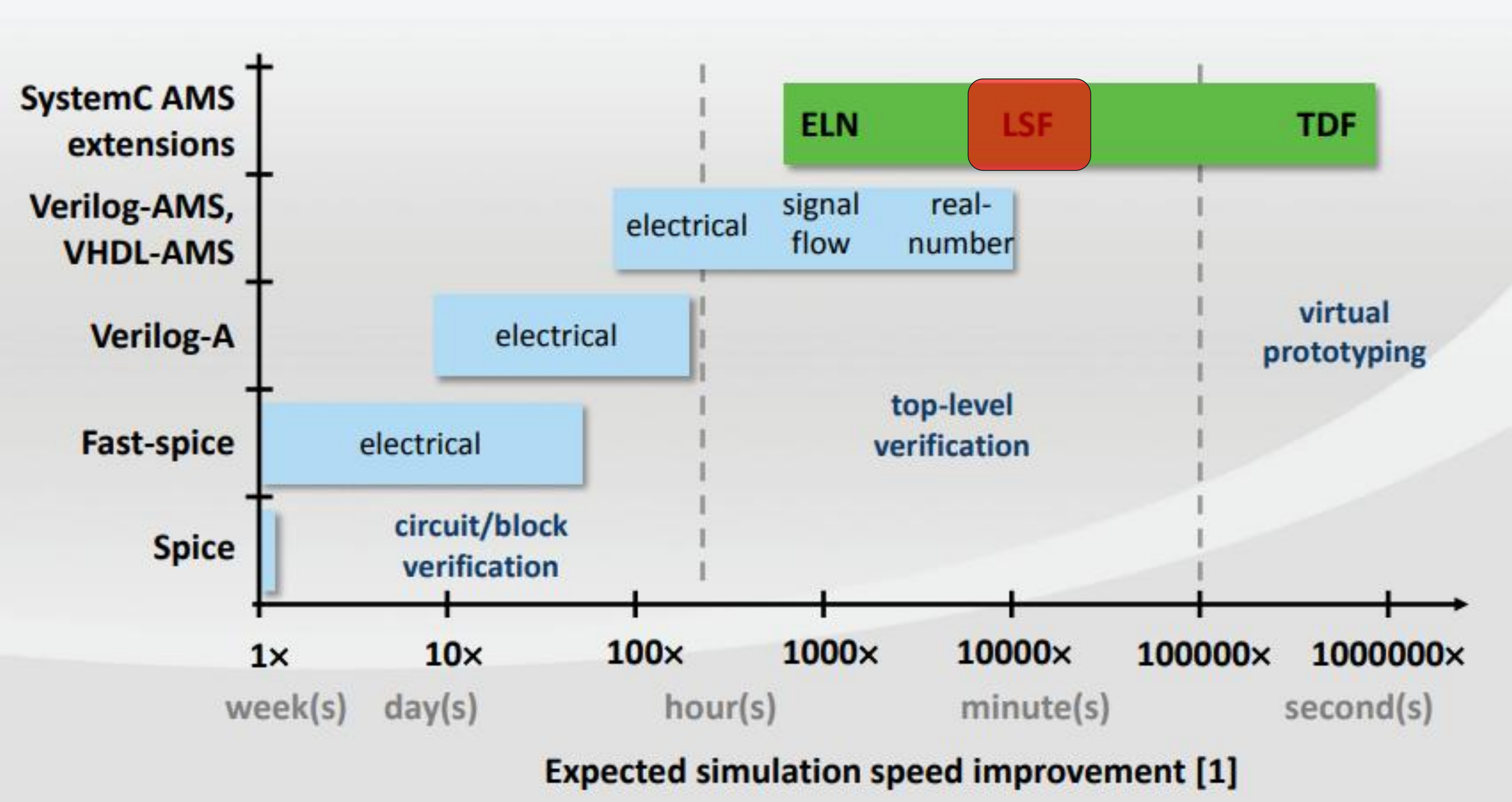
```cpp
#include <systemc-ams.h>
#include "lowpassFilter.h"
#include "squareWave.h"

int sc_main(int argc, char* argv[]){

    sc_set_time_resolution(10.0, SC_NS);
    sc_signal <double> va, vb;

    lowpassFilter filter ("fil", 100, 100.0e-6);
    squareWave square ("SW");

    square.out(va);
    filter.in(va);
    filter.out(vb);


    sc_trace_file* tr = sc_create_vcd_trace_file("tr");
//      sca_util::sca_trace_file* atf = sca_util::sca_create_tabular_trace_file( "my_trace.da

    sc_trace(tr, va ,"va");
    sc_trace(tr, vb ,"vb");
    sc_start(400, SC_MS);
    return 0;
}
```


Signals
Waves
Time
va =1
vb =0.42
100 ms
200 ms
300 ms

ELN execution semantics



Linear Signal Flow Modeling



Linear Signal Flow Modeling

- The Linear Signal Flow model of computation allows the modeling of AMS behavior defined as relations between variables of a set of linear algebraic equations

- Signal flow models can be described in a block diagram notation.

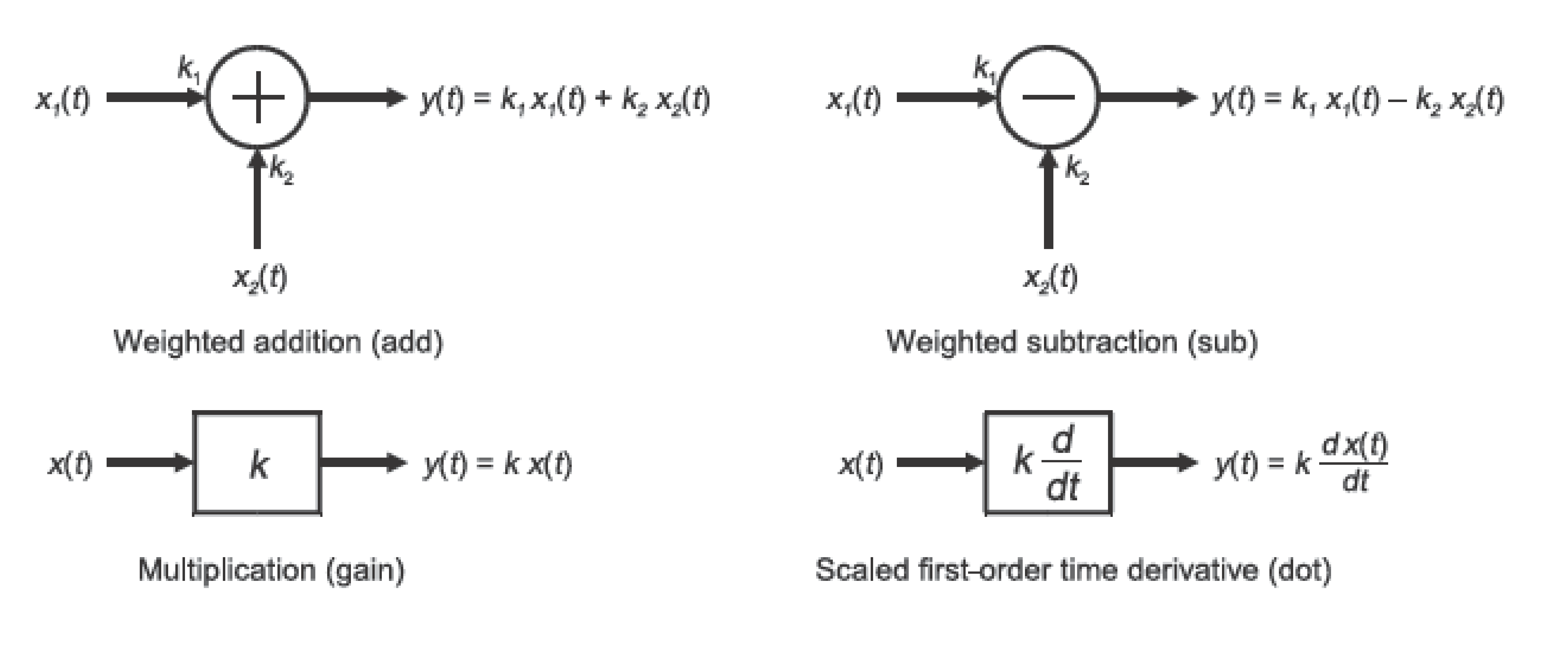
- The elementary parts or functions are represented by blocks.

- Signals are used to interconnect these blocks.

- The resulting relations between the blocks define equivalent mathematical equations.

Setup of LSF equation system

- The SystemC AMS extensions offer a finite set of predefined LSF primitive modules implementing functions such as addition, multiplication, integration, etc.



LSF Modeling Primitive Modules

A.5.4. sca_lsf::sca_dot

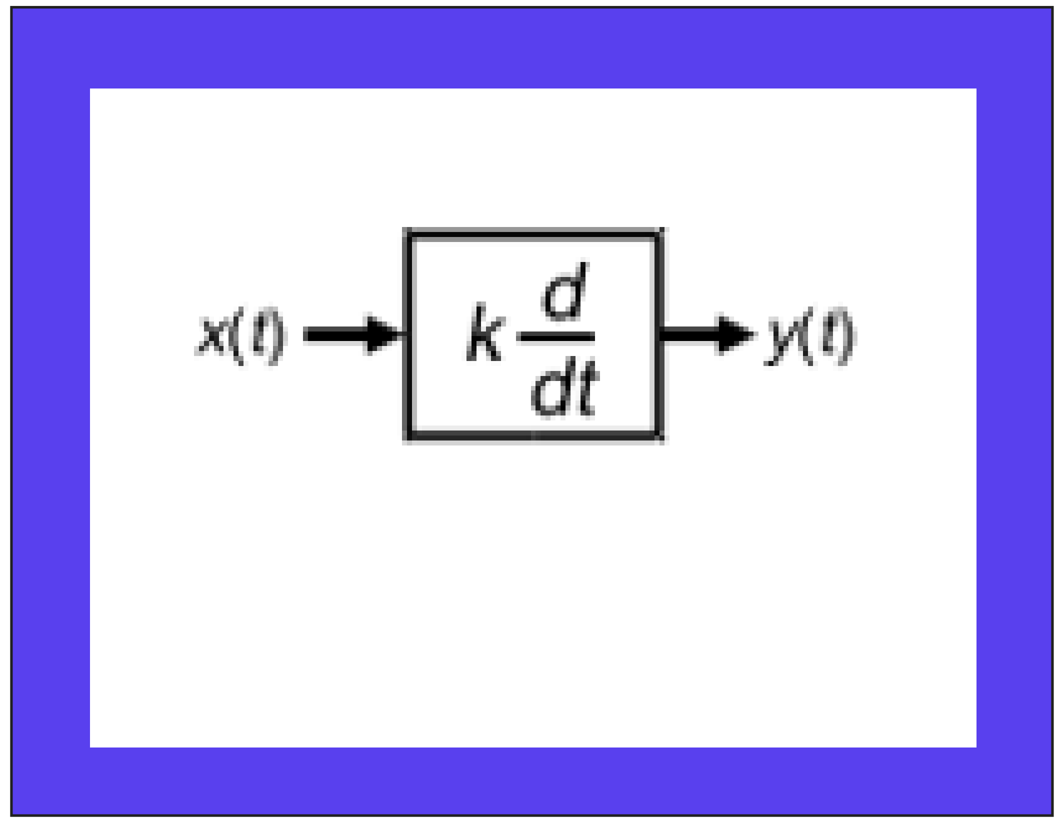
Description

Scaled first-order time derivative of an LSF signal.

Definition

Sca_lsf::sca_dot(nm, k)

Symbol



Equation

$$y(t) = k \cdot \frac{dx(t)}{dt}$$

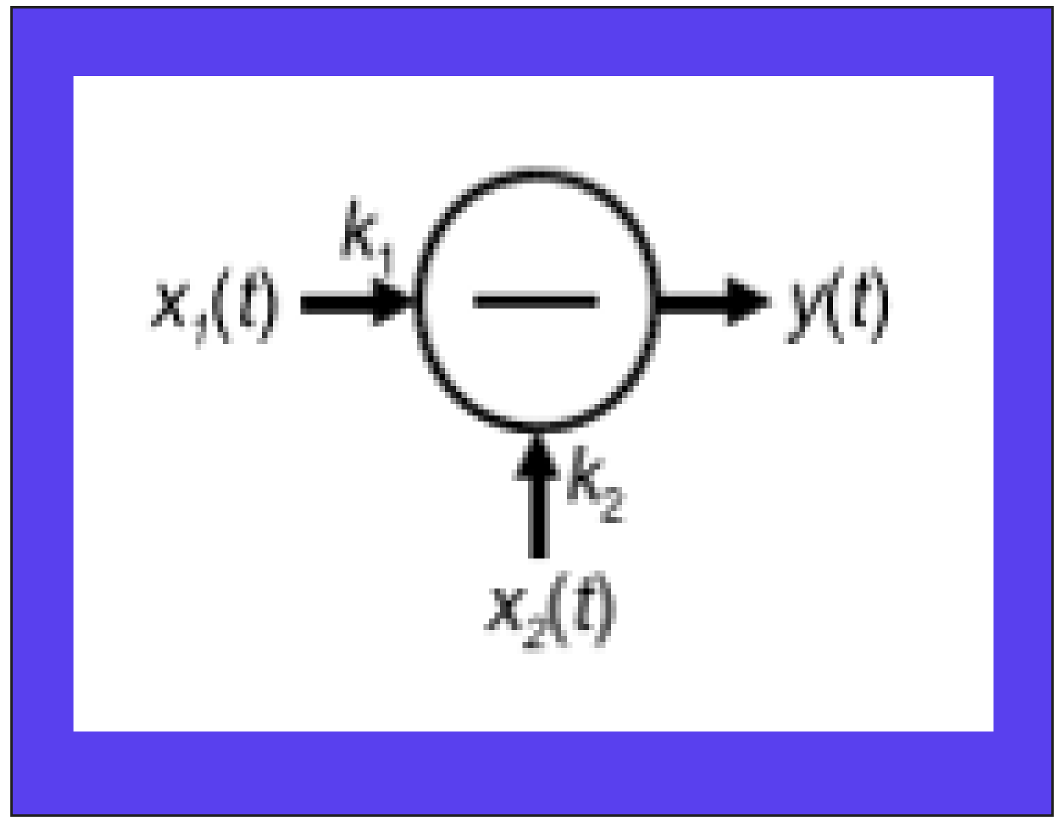
Parameters

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
k	double	1.0	Scale coefficient

LSF Modeling Primitive Modules

Sca_lsf::sca_sub(nm, k1,k2)

Symbol



Equation

$$y(t) = k_1 \cdot x_1(t) - k_2 \cdot x_2(t)$$

Parameters

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
k1	double	1.0	Weighting coefficient for LSF signal at port x1
k2	double	1.0	Weighting coefficient for LSF signal at port x2

LSF Modeling Primitive Modules

LSF module name	Description
sca_lsf::sca_add	Weighted addition of two LSF signals.
sca_lsf::sca_sub	Weighted subtraction of two LSF signals.
sca_lsf::sca_gain	Multiplication of an LSF signal by a constant gain.
sca_lsf::sca_dot	Scaled first-order time derivative of an LSF signal.
sca_lsf::sca_integ	Scaled time-domain integration of an LSF signal.
sca_lsf::sca_delay	Scaled time-delayed version of an LSF signal.
sca_lsf::sca_source	LSF source.
sca_lsf::sca_ltf_nd	Scaled Laplace transfer function in the time-domain in the numerator-denominator form.
sca_lsf::sca_ltf_zp	Scaled Laplace transfer function in the time-domain in the zero-pole form.
sca_lsf::sca_ss	Single-input single-output state-space equation.
sca_lsf::sca_tdf::sca_gain, sca_lsf::sca_tdf_gain	Scaled multiplication of a TDF input signal with an LSF input signal.
sca_lsf::sca_tdf::sca_source, sca_lsf::sca_tdf_source	Scaled conversion of a TDF input signal to an LSF output signal.
sca_lsf::sca_tdf::sca_sink, sca_lsf::sca_tdf_sink	Scaled conversion from an LSF input signal to a TDF output signal.
sca_lsf::sca_tdf::sca_mux, sca_lsf::sca_tdf_mux	Selection of one of two LSF input signals by a TDF control signal (multiplexer).

LSF Modeling Primitive Modules

LSF module name	Description
sca_lsf::sca_tdf::sca_demux, sca_lsf::sca_tdf_demux	Routing of an LSF input signal to either one of two LSF output signals controlled by a TDF signal (demultiplexer).
sca_lsf::sca_de::sca_gain, sca_lsf::sca_de_gain	Scaled multiplication of a discrete-event input signal by an LSF input signal.
sca_lsf::sca_de::sca_source, sca_lsf::sca_de_source	Scaled conversion of a discrete-event input signal to an LSF output signal.
sca_lsf::sca_de::sca_sink, sca_lsf::sca_de_sink	Scaled conversion from an LSF input signal to a discrete-event output signal.
sca_lsf::sca_de::sca_mux, sca_lsf::sca_de_mux	Selection of one of two LSF input signals by a discrete-event control signal (multiplexer).
sca_lsf::sca_de::sca_demux, sca_lsf::sca_de_demux	Routing of an LSF input signal to either one of two LSF output signals controlled by a discrete-event signal (demultiplexer).

LSF Ports

- There are two classes of LSF ports:
- LSF input ports of class sca_lsf::sca_in.
- LSF output ports of class sca_lsf::sca_out.

```
SC_MODULE ( filter ) {
   public :
   sca_lsf :: sca_in in ;        // input port
   sca_lsf :: sca_out out ;      // output port

   sca_lsf :: sca_signal sig ; // internal signal
```

- Unlike TDF ports, the LSF ports do not provide member functions to directly read to or write from the channel.

LSF Signals

- LSF signals are used to connect LSF primitive modules together.
- LSF signals carry the continuous-time and continuous-value of a signal
- LSF ports determine the direction of the signals from one LSF module to another.

Modeling Continuous-Time Behavior

- LSF models can be used to implement linear dynamic, continuous-time behavior.

- LSF models can only be composed using LSF primitive modules.

- LSF model is always a structural model.

Structural composition of LSF modules

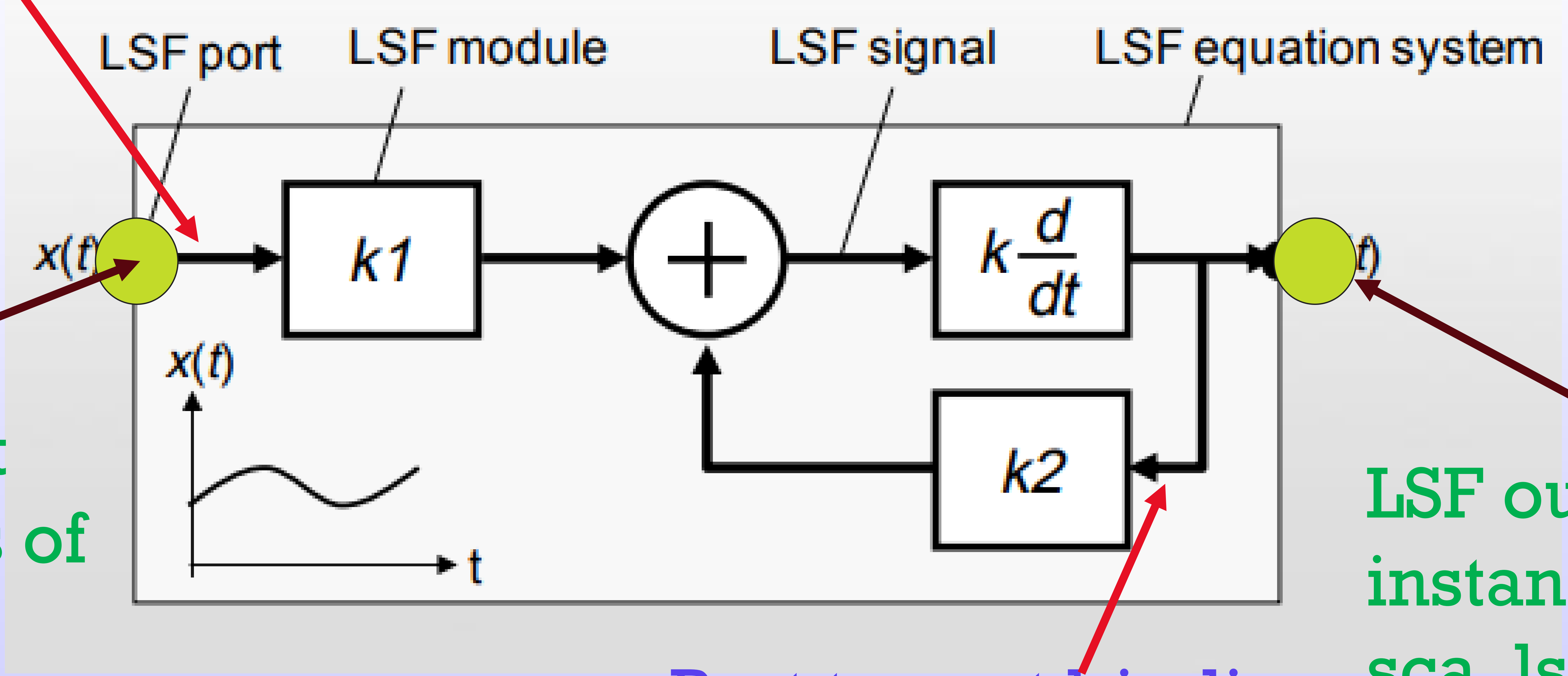
- LSF modules should be instantiated as child modules inside a regular SystemC parent module

- The parent module also instantiates all necessary ports to communicate with the outside world and internal signals for the interconnection of the child modules.

- The parameterization of the instantiated modules as well as the interconnection of the modules should be done in the constructor

LSF port binding

Port to port binding



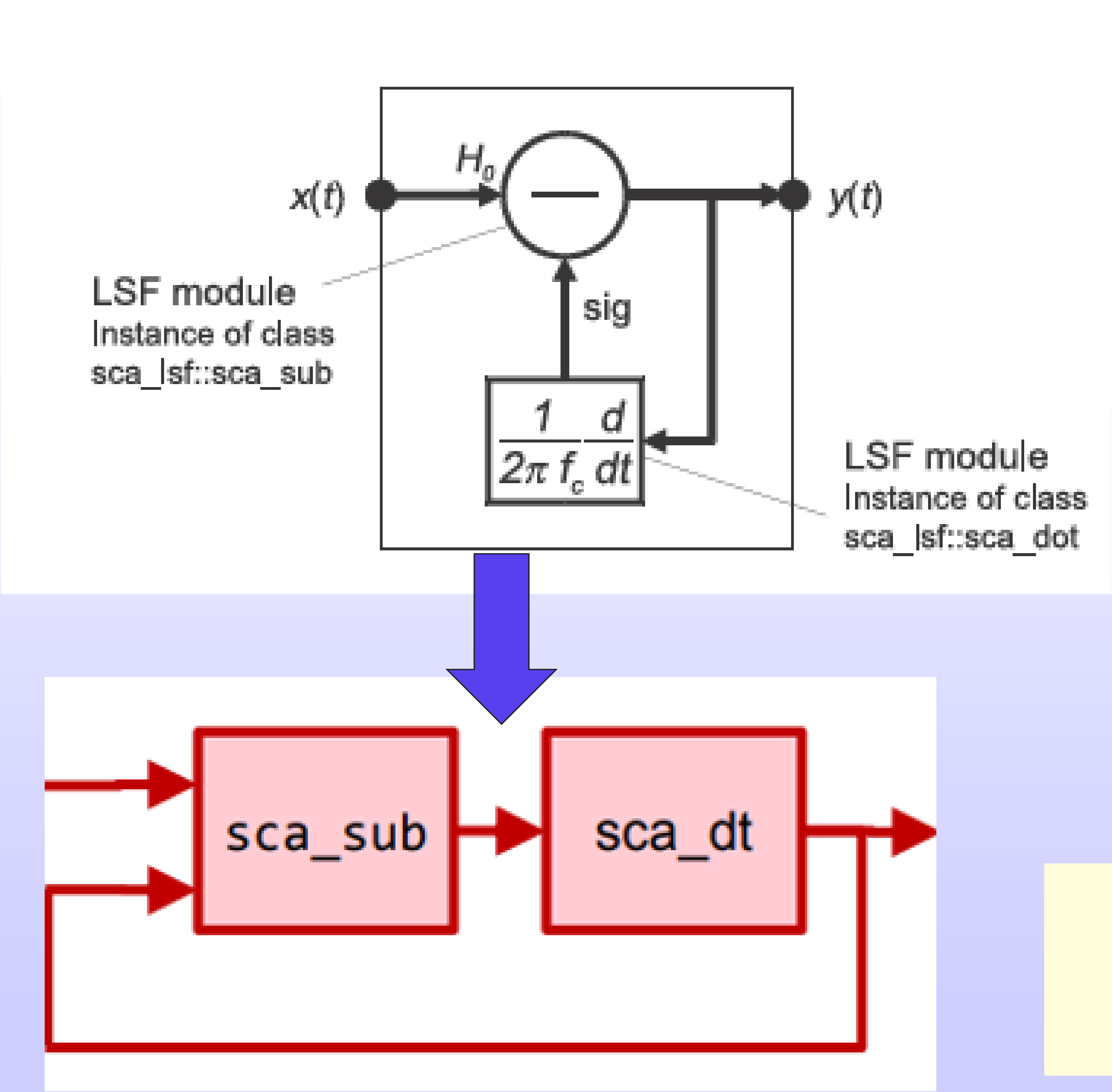
LSF input port instance class of sca_lsf:sca_in

LSF output port instance class of sca_lsf:sca_out

Port to port binding

SystemC parent module Object of class sc_core::sc_module

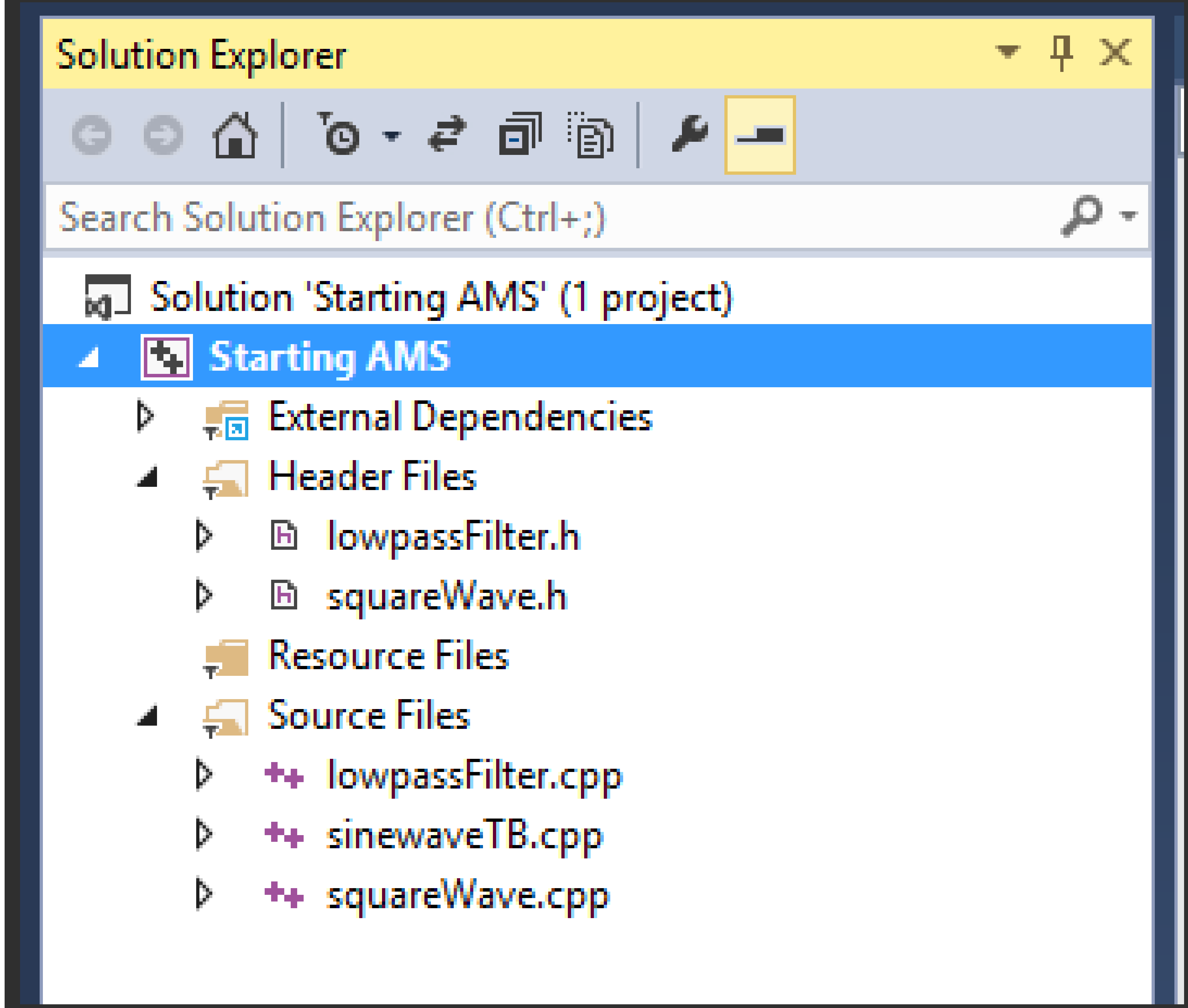
LSF Modeling Primitive Modules



$$H(s) = \frac{H_0}{1 + \frac{1}{2\pi f_c} s}$$

$$y(t) = H_0 x(t) - \frac{1}{2\pi f_c} \frac{dy(t)}{dt}$$

Example
Low Pass Filter

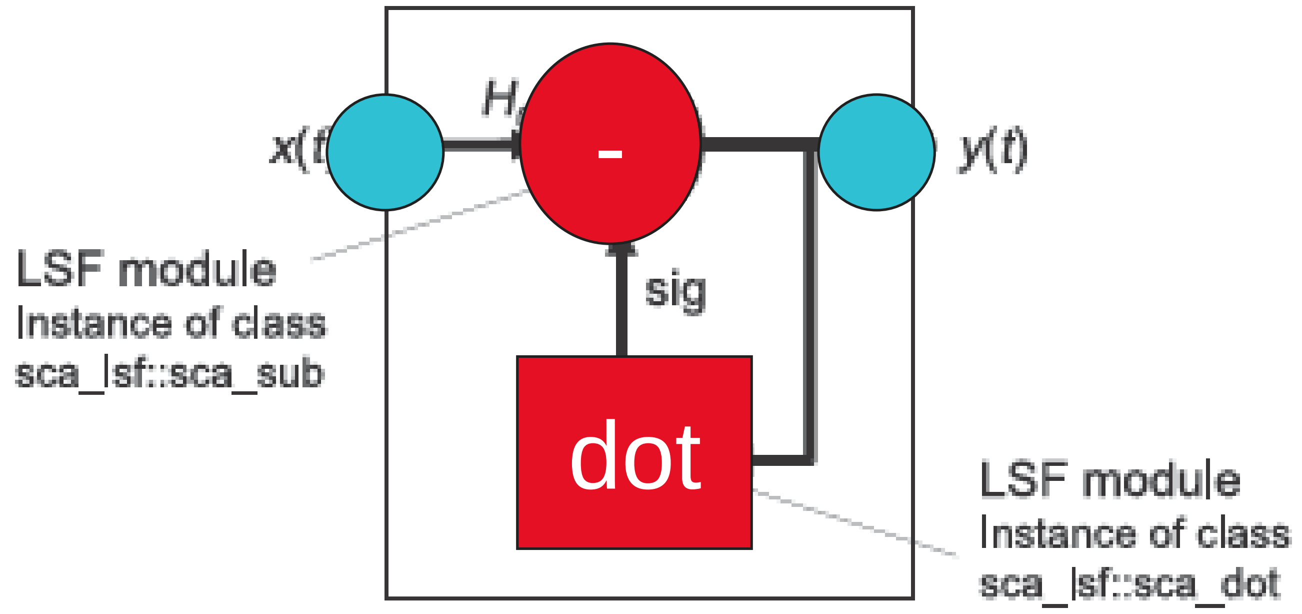


LSF example

lowpassFilter.h × squareWave.h squareWave.cpp lowpassFilter.cpp

(Global Scope)

```
#include <systemc.h>
#include <systemc-ams.h>

SC_MODULE(lowpassFilter){
    // ports
    sc_in <double> in;
    sc_out  <double> out;

    // primitive module instantiations
    sca_lsf::sca_de::sca_source vin;
    sca_lsf::sca_de::sca_sink vout;

    sca_lsf::sca_sub sub1;
    sca_lsf::sca_dot dot1;

    SC_HAS_PROCESS(sc_module_name);
    lowpassFilter(sc_core::sc_module_name, double h0 = 1.0, double fc = 2.0e3);

private:
        sca_lsf::sca_signal x1, sig, x3;
};
```

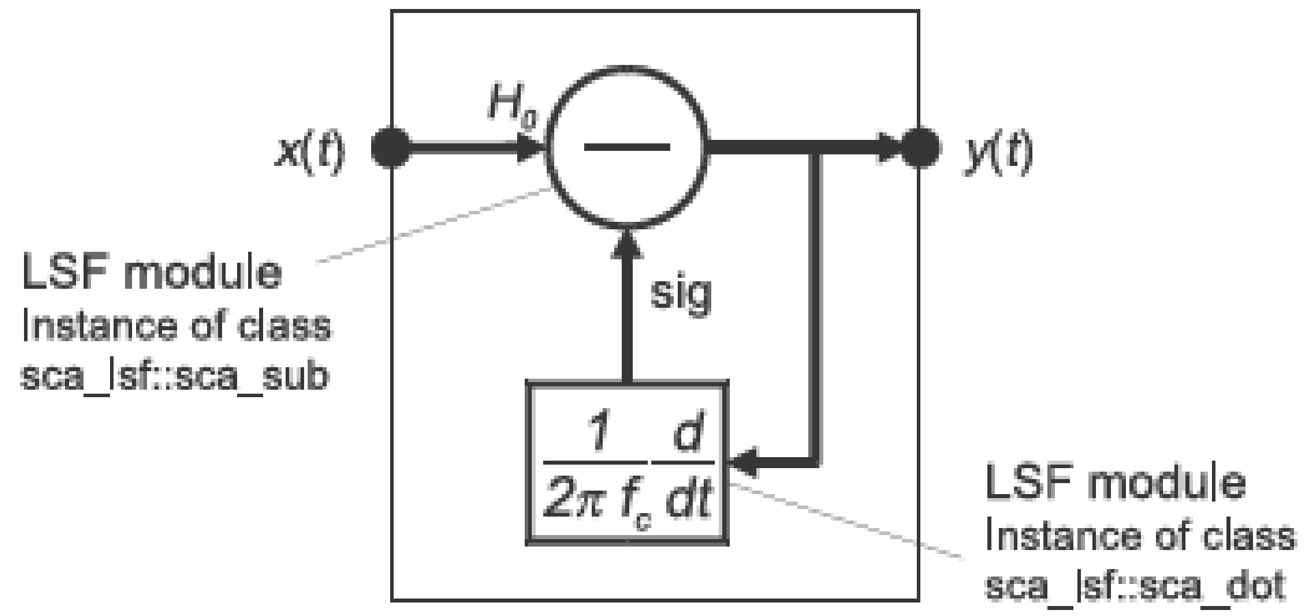


LSF example

lowpassFilter.h squareWave.h squareWave.cpp lowpassFilter.cpp

lowpassFilter lowpassFilter(sc_core::sc_module

```
#include "lowpassFilter.h"

lowpassFilter::lowpassFilter (sc_core::sc_module_name, double h0 = 1.0, double fc = 2.0e3)

    :sub1("sub1", h0), dot1("dot1", 1.0/(1.0*3.14*fc)), vin("vin"), vout("vout")
    {

        vin.inp(in);
        vin.y(x1);
        vin.set_timestep(1, SC_MS);

        sub1.x1(x1);
        sub1.x2(sig);
        sub1.y(x3);

        dot1.x(x3);
        dot1.y(sig);


        vout.x(x3);
        vout.outp(out);

    }
```



LSF example

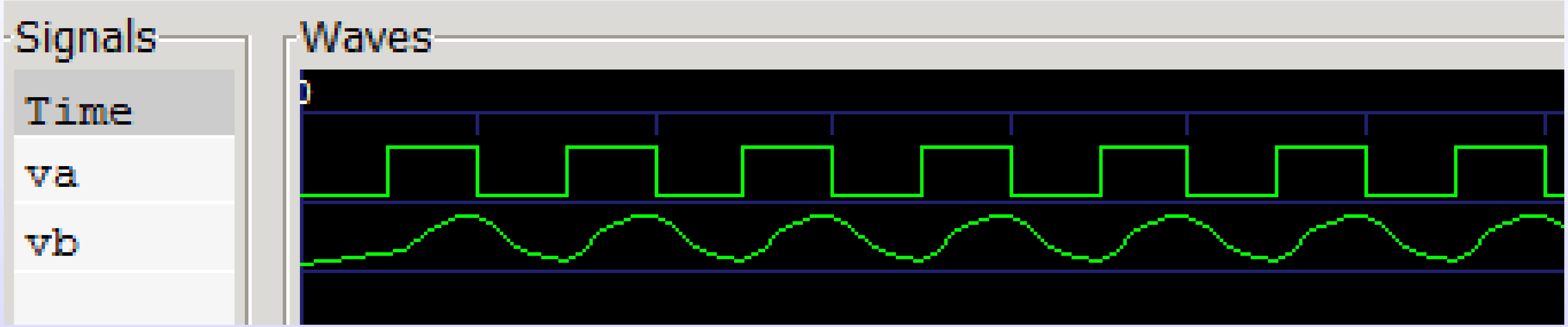
squareWave.h

```cpp
#include <systemc-ams.h>

SC_MODULE(squareWave){
    sc_out <double> out;

    SC_CTOR(squareWave){
        SC_THREAD(wave);
    }
    void wave();
};
```

squareWave.cpp

```cpp
#include "squareWave.h"

void squareWave::wave()
{
    while (1){
        wait(25, SC_MS);
        out->write(1.0);
        wait(25, SC_MS);
        out->write(0.0);
    }
}
```

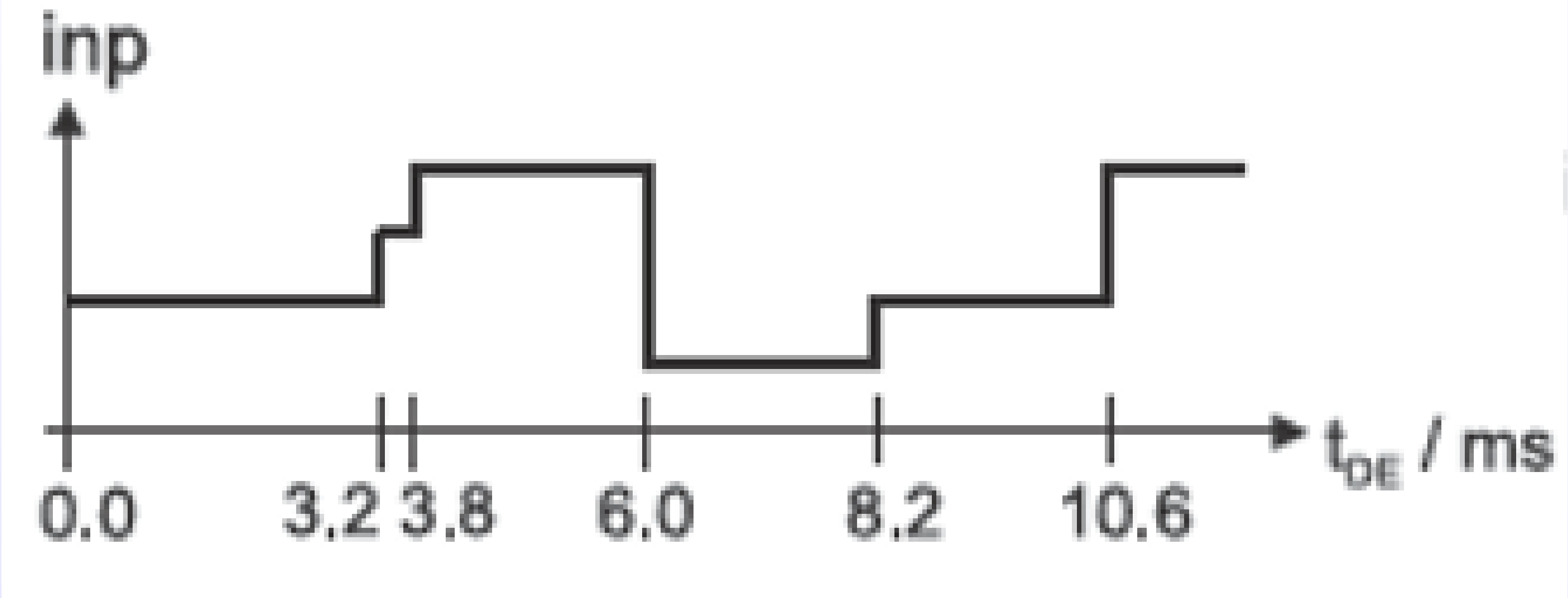
LSF example

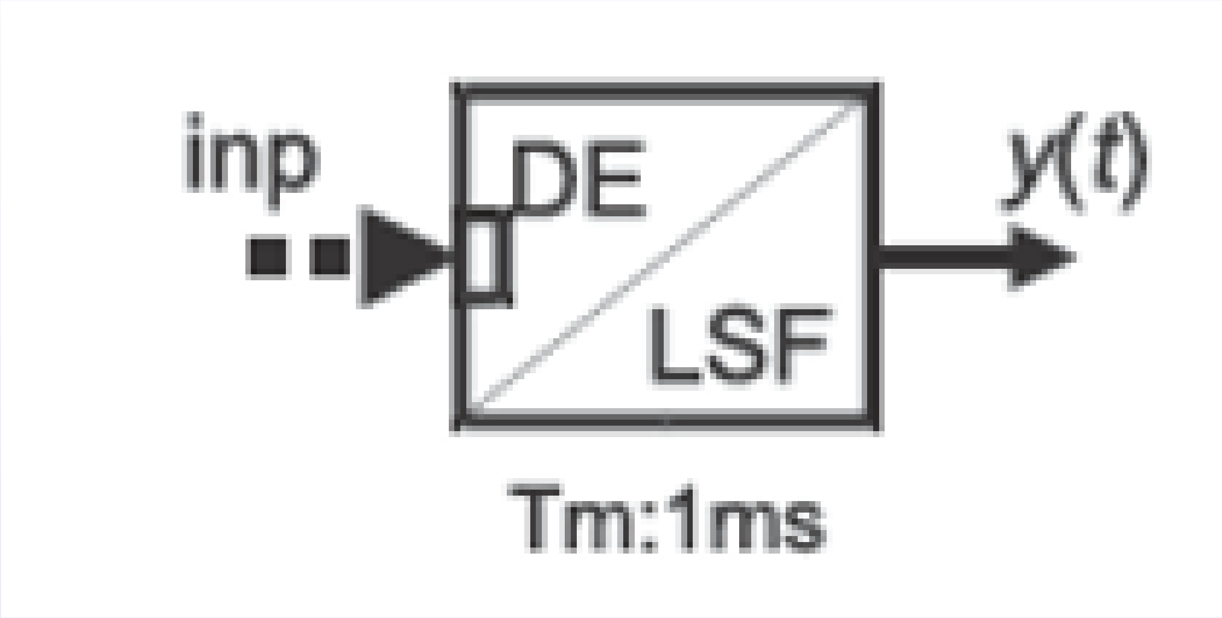


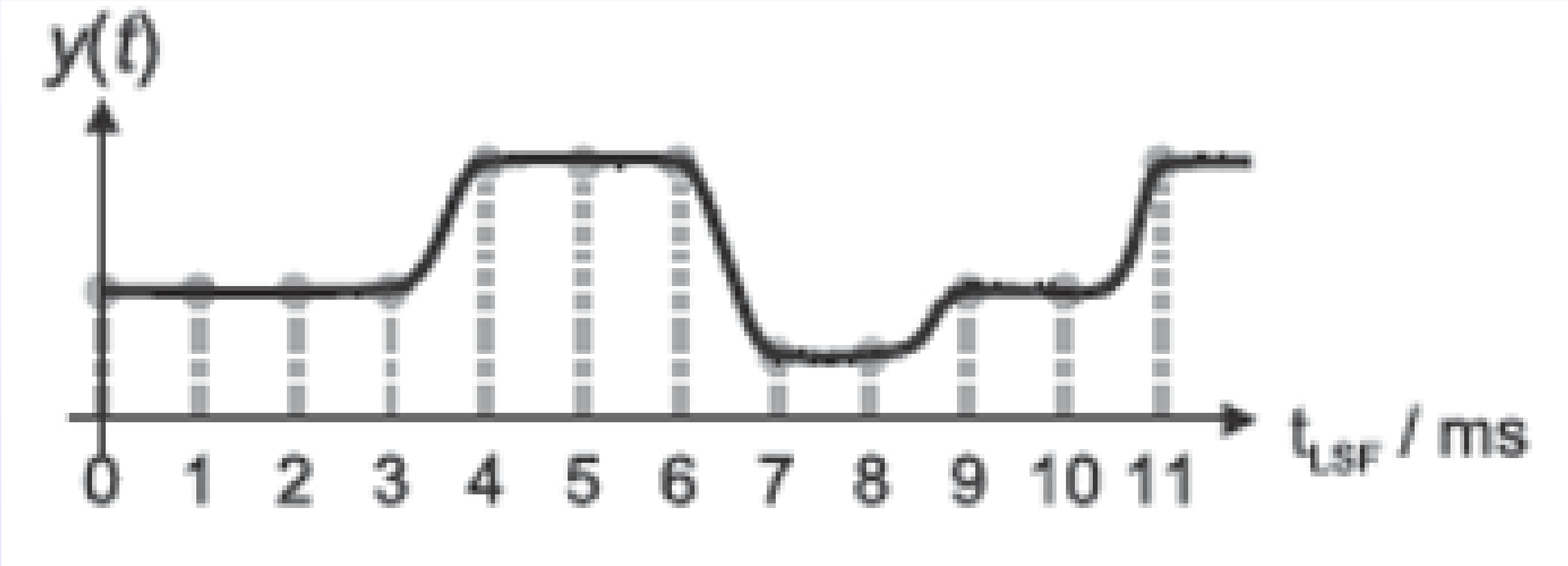
Interaction between LSF and discrete-event or TDF models

- Specialized LSF primitive modules with ports to the discrete-event domain and TDF models of computation are available, which are called converter modules.
- Main purpose of these modules is to establish an interface to convert and transfer data from one model of computation to the other.

Reading from discrete-event models

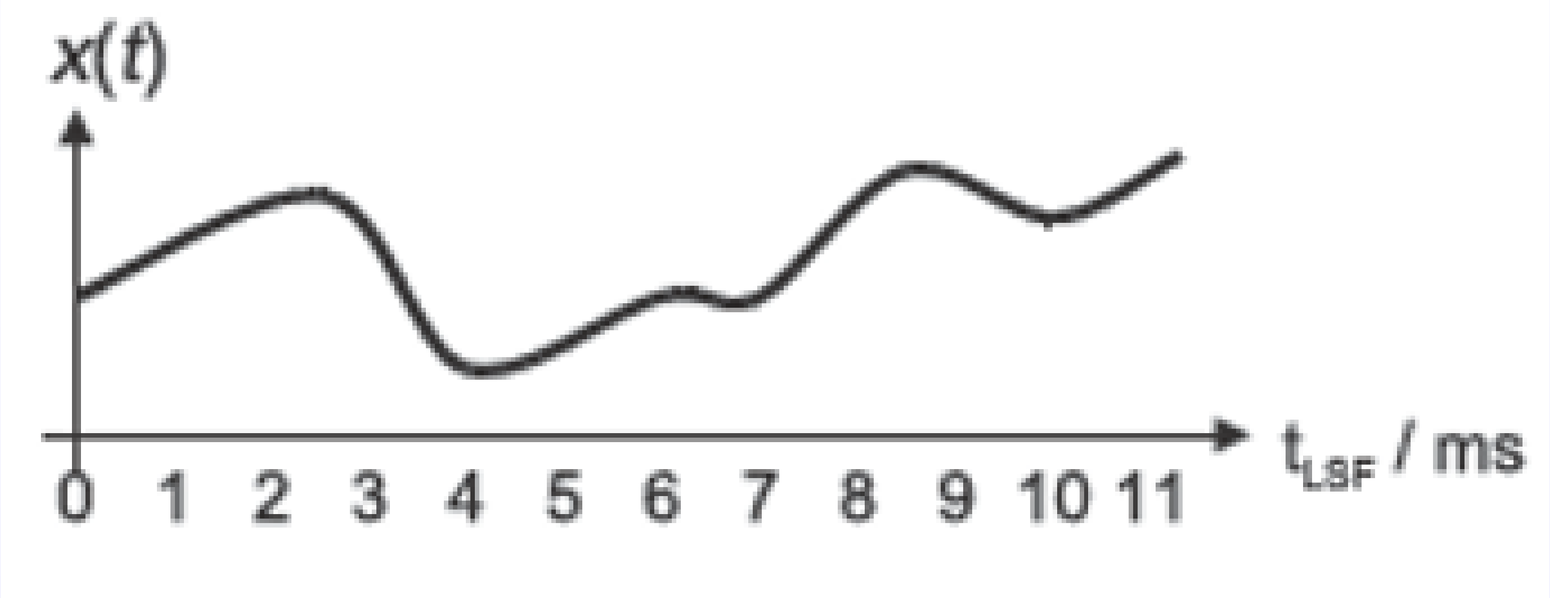


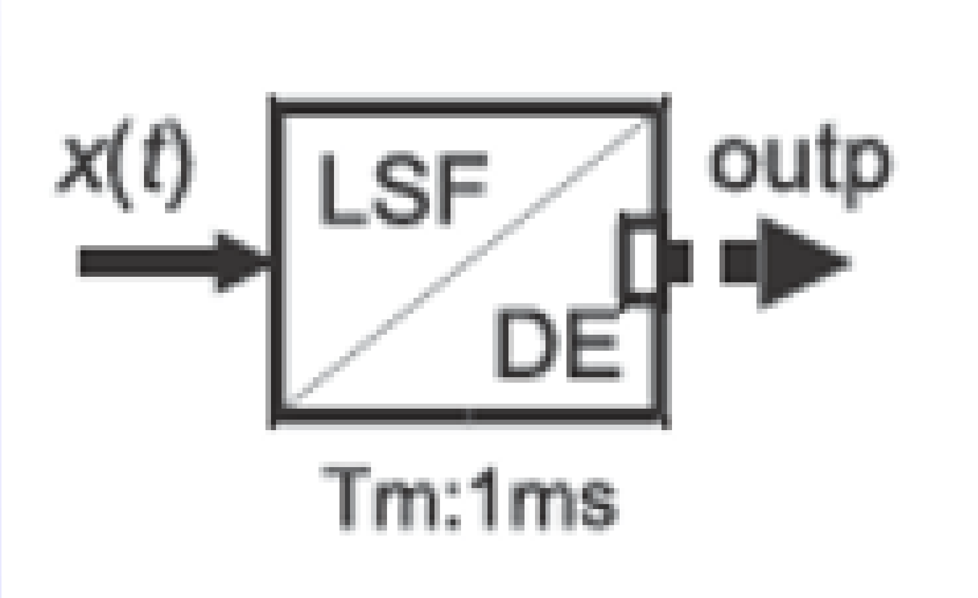




Discrete-event signal Instance of class sc_core::sc_signal<double>	LSF converter module Instance of class sca_lsf::sca_de::sca_source	LSF signal Instance of class sca_lsf::sca_signal

Writing to discrete-event models



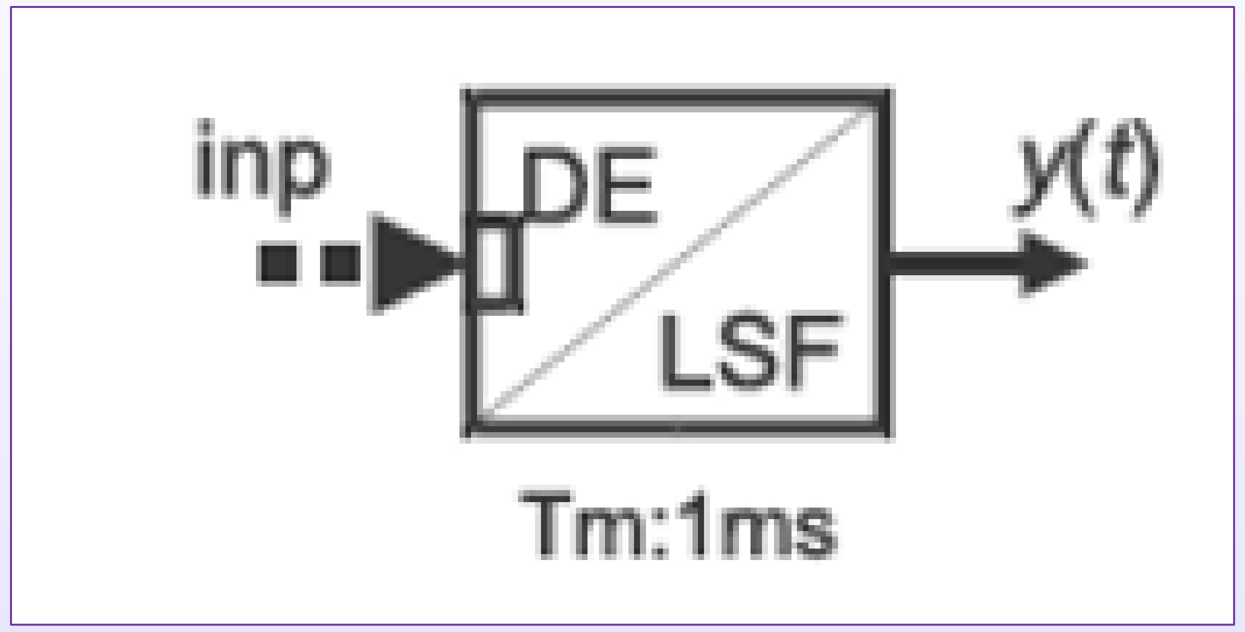


LSF signal
Instance of class
sca_lsf::sca_signal

LSF converter module
Instance of class
sca_lsf::sca_de::sca_sink

Discrete-event signal
Instance of class
sc_core::sc_signal<double>
or
sc_core::sc_buffer<double>

```
sca_lsf::sca_de::sca_source( nm, scale);

sca_lsf::sca_de_source( nm, scale);
```



Symbol

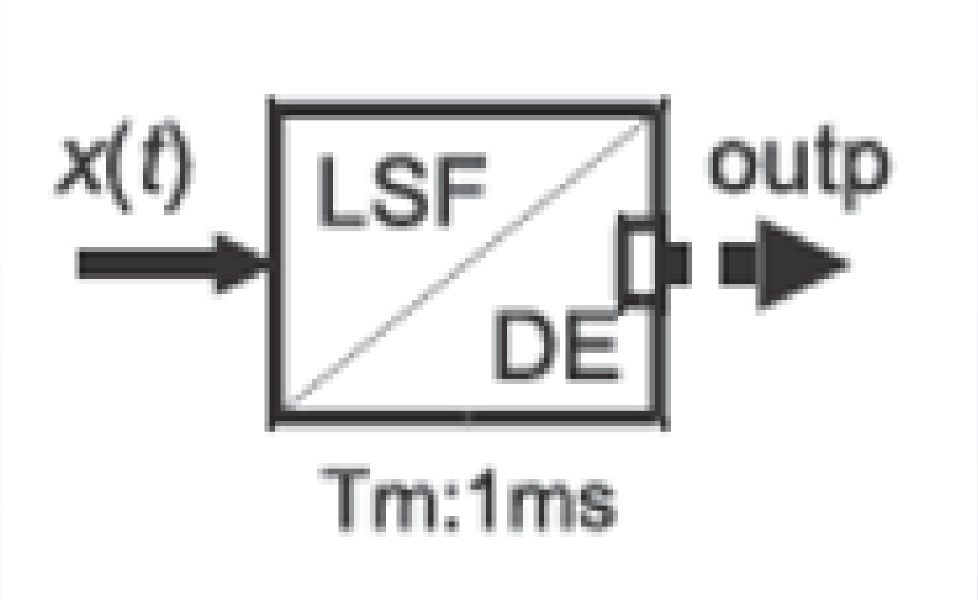
Equation:

$$y(t) = scale.inp$$

Parameters:

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name
scale	Double	1.0	Scale coefficient

```
sca_lsf::sca_de::sca_sink( nm, scale);

sca_lsf::sca_de_sink( nm, scale);
```



Symbol

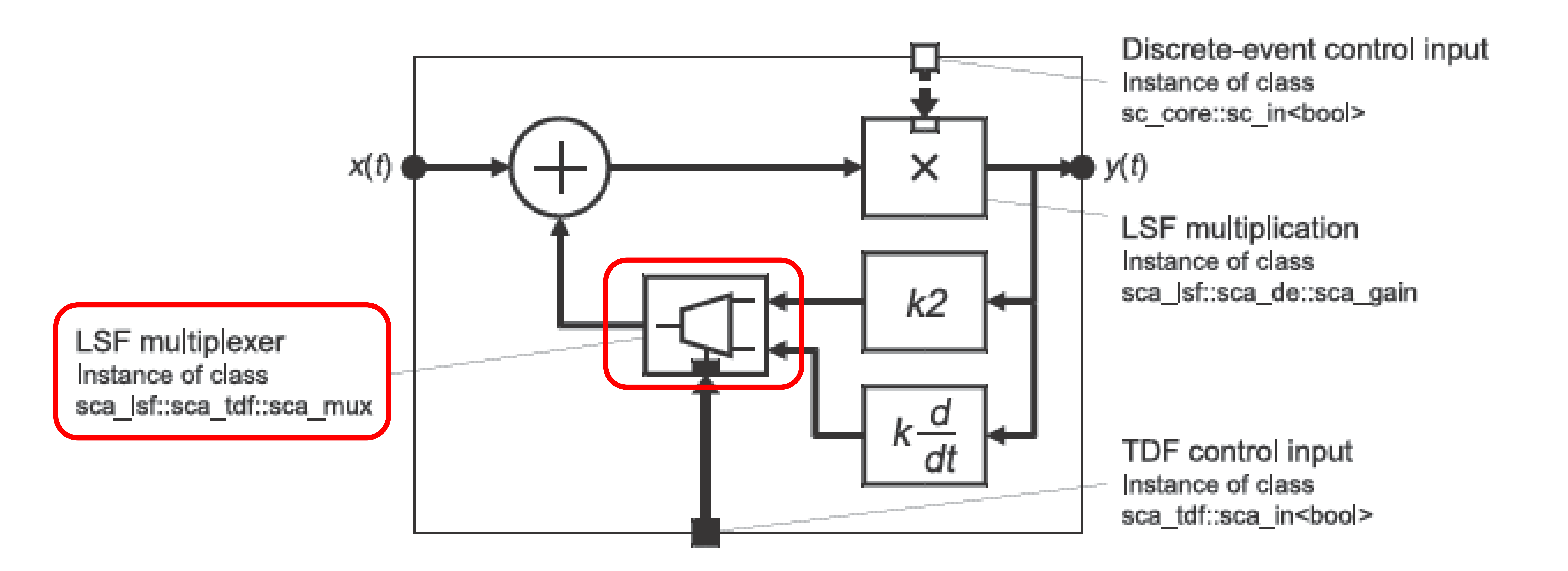
Equation:

There is no equation contributed to overall equation system for this module.

Parameters:

Name	Type	Default	Description
nm	sc_core::sc_module_name		Module name

Using discrete-event or TDF control signals



LSF execution semantics

LSF time step calculation and propagation:
Define time step and check consistency

↓

LSF equation set-up and solvability check:
Define the equation system and check if it can be solved

↓

LSF initialization:
Set initial conditions, e.g., defined in LSF primitives

↓

LSF time-domain simulation:
Provide results at the calculated time points

LSF elaboration phase (first two steps)

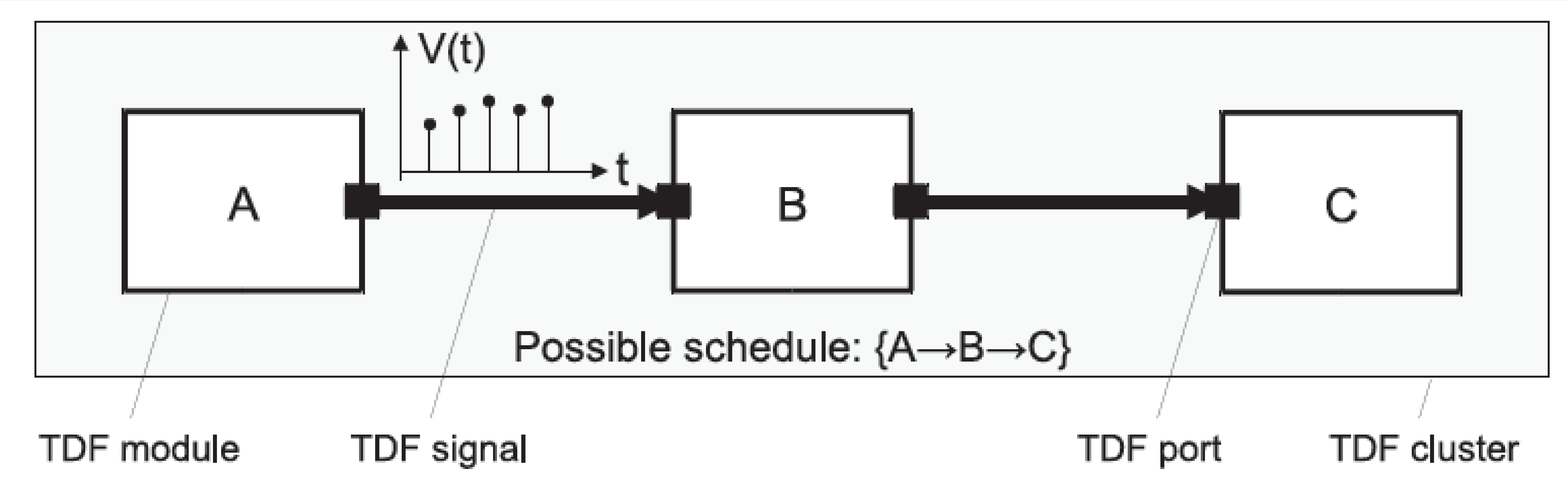
LSF simulation phase (last two steps)

Timed Data Flow(TDF)

- Discrete-time modeling and simulation
- By considering data as signals sampled in time
- Static scheduling (not dynamic)

TDF Model

- A TDF model:
  - A set of connected TDF modules
  - Forming a directed graph called TDF cluster
    - TDF modules : vertices
    - TDF signals : edges



TDF Model

- A given function is processed if and only if there are enough samples available at the input ports of its module
- Number of samples read from or written to the module ports is fixed during simulation
- The fixed interval between two samples is called time step.

TDF Module and Port Attributes

- The flexibility and expressiveness of TDF modeling comes from the ability to define the attributes of each TDF module and of each of its ports
- The order of activation of the TDF modules in a cluster and the number of samples they read (consume) and write (produce) can be statically determined before simulation starts
- A TDF cluster can be defined as the set of connected TDF modules, which belong to the same static schedule

TDF Module and Port Attributes

Attributes

- Rate

Port attribute – number of samples for reading / writing during one module execution

- Delay

Port attribute – delay, number of samples to be inserted while initializing. Will be read before an actual module produced sample.

- Timestep

Port and module attribute – time distance between two samples or two module activations.

Language constructs (TDF modules)

```
SCA_TDF_MODULE(mytdfmodel)           // create your own TDF primitive module
{
  sca_tdf::sca_in<double>  in1, in2;  // TDF input ports
  sca_tdf::sca_out<double> out;       // TDF output port

  void set_attributes()
  {
    // placeholder for simulation attributes
    // e.g. time step between module activations
  }

  void initialize()
  {
    // put your initial values here
  }

  void processing()
  {
    // put your signal processing or algorithm here
  }

  SCA_CTOR(mytdfmodel) {}
};
```

No hierarchy

Timed semantics

Describes computation

Language constructs (TDF modules)

```
SCA_TDF_MODULE(my_tdf_module)
{
// port declarations
sca_tdf::sca_in<double> in;
sca_tdf::sca_out<double> out;
SCA_CTOR(my_tdf_module) {}
... //Functions declaration
};
```

TDF Modules-Module Attributes

- The set_attributes function is used for defining module attributes

```
void set_attributes()
{
 set_timestep(10.0, sc_core::SC_MS); // module time step assignment of a of 10 ms

 out.set_delay(1); // set  1 delay to port out
}
```

TDF Modules -Module Initialization

- The member function initialize can be used for:
  - Setting local variables used as state variables
  - Reading port or module attributes such as time steps or port rates
  - Initializing ports with a delay
- This member function is executed only once, just before the actual module activation starts

```
void initialize()
{
        s = 4.56;
        std::cout << out.name() << ": Time step = " << out.get_timestep() ;
        out.initialize(1.23);
}
```

TDF Modules-Module Activation

- The member function processing is the only mandatory function that needs to be overloaded in any TDF module
- It defines the discrete-time or continuous-time behavior of the TDF module
- Is executed at each module activation

```
void processing()
{
      out.write(val); // writes value to output port out
}
```

TDF Modules-Module Constructor

- The macro SCA_CTOR helps to define the standard constructor of a module of class sca_tdf::sca_module
- It has module name as its mandatory argument
- Use a regular constructor for more arguments

```
my_tdf_module( sc_core::sc_module_name nm, double param_ )
: param(param_) {}
```

TDF Modules-Constraint on Usage

- The member functions set_attributes, initialize, processing, and ac_processing should not be called directly by the user
- SystemC member functions and macros like SC_HAS_PROCESS, SC_METHOD, SC_THREAD, wait, next_trigger, sensitive should not be used in a TDF module
- The function sc_core::sc_time_stamp should not be used inside a TDF module, instead, the member function get_time should be used

TDF Ports

- There are currently four classes of TDF ports:
  - TDF ports
    - sca_tdf::sca_in<T> (input port)
    - sca_tdf::sca_out<T> (output port)
  - TDF converter ports
    - sca_tdf::sca_de::sca_in<T> (input converter port)
    - sca_tdf::sca_de::sca_out<T> (output converter port)

TDF Ports

```
SCA_TDF_MODULE(my_tdf_module)
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;
    sca_tdf::sca_de::sca_in<bool> inp;
    sca_tdf::sca_de::sca_out< sc_dt::sc_logic > outp;
        // rest of module not shown
};
```

TDF Ports-Port Attributes

```
void set_attributes()
{
 out.set_timestep(0.01, sc_core::SC_US); // set time step of port out
 out.set_rate(1); // set rate of port out to 1
 out.set_delay(2); // set delay of port out to 2 samples
 outp.set_timeoffset(0.2, sc_core::SC_US);
 // set absolute time of first sample of converter port
}
void initialize()
{
 out.get_rate(); // return the rate of port out
 out.get_delay(); // return the delay of port out
 out.get_timestep(); // return actual timestep of port out
 outp.get_timestep(); // return actual timestep of converter port outp
 outp.get_timeoffset(); // return absolute time of first sample of converter port outp
}
```

TDF Ports-Port Initialization

```
void initialize() // use initialize method of TDM module to initialize ports
{
      // initialize port out (which has a delay attribute of 2)
      out.initialize(1.23); // initialize first sample with value 1.23 or
      out.initialize(1.23,0); // initialize first sample with value 1.23
      out.initialize(4.56,1); // initialize second sample with value 4.56
}
```

TDF Ports-Port Read and Write Access

- Single rate TDF input port

```
SCA_TDF_MODULE(my_tdf_sink)
{
  sca_tdf::sca_in<double> in;
  SCA_CTOR(my_tdf_sink) : in("in") {}
  void processing()
  {
        // local variable
   double val; // variable to store value read from port in
   val = in.read(); // reading first sample from the input port
  }
};
```

Multirate TDF input port

```
SCA_TDF_MODULE(my_multi_rate_sink)
{
  sca_tdf::sca_in<double> in;
  SCA_CTOR(my_multi_rate_sink) : in("in") {}

  void set_attributes()
  {
    in.set_rate(2); // 2 samples read per module activation
  }
  void processing()
  {
    // local variable
    double val; // variable to store values read from port in
    val = in.read(); // read first sample
    val = in.read(0); // same method with index for first sample
    val = in.read(1); // same method with index for second sample
  }
};
```

Single rate TDF output port

```
SCA_TDF_MODULE(my_const_source)
{
 sca_tdf::sca_out<double> out;
 my_const_source( sc_core::sc_module_name, double val_ = 1.0 )
 : out("out"), val( val_ ) {}
 void processing()
 {
  out.write( val ); // writes val as a new sample to the port out
 }
 private:
 double val; // value to be written to the port out
};
```

Multirate TDF output port

```
SCA_TDF_MODULE(my_multi_rate_const_source)
{
 sca_tdf::sca_out<double> out;
 my_multi_rate_const_source(sc_core::sc_module_name, double val_ = 1.0 )
 : out("out"), val( val_ ) {}
 void set_attributes()
 {
  out.set_rate(2); // 2 samples written per module activation
 }
 void processing()
 {
  out.write(val); // writes val as the first sample to the port out
  out.write(val,0); // writes val as the first sample to the port out by specifying the index 0
  out.write(val,1); // writes val as the second sample to the port out by specifying the index 1
 }
 private:
 double val; // value to be written to the port out
};
```

A simple TDF example – sine wave

sine.cpp

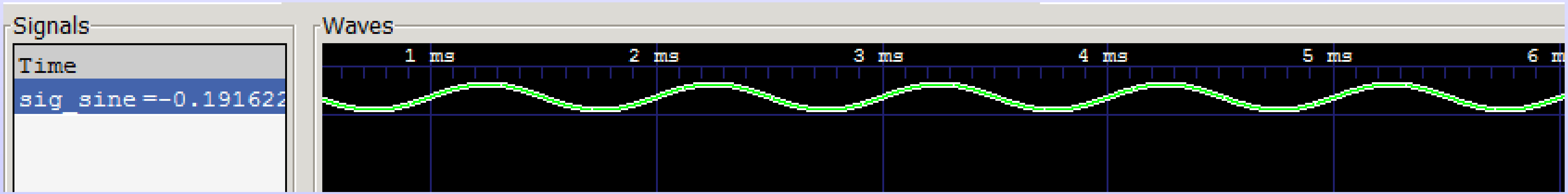
sine | (Global Scope) | sc_main(int argc, char * argv[])

```
#include "systemc-ams.h"
SCA_TDF_MODULE(sine) {
    sca_tdf::sca_out<double> out; // output port

    sine(sc_core::sc_module_name nn, double ampl_ = 1.0,
        sca_core::sca_time Tm_ = sca_core::sca_time(100, sc_core::SC_NS))
        : Tm(Tm_) {}


    void set_attributes()
    {
        set_timestep(Tm);
    }

    void processing() {
        out.write(sin(sc_time_stamp().to_seconds()*(1000.*2.*3.14)));
    }

private:
    double ampl; // amplitude
    sca_core::sca_time Tm; // module time step
};
```

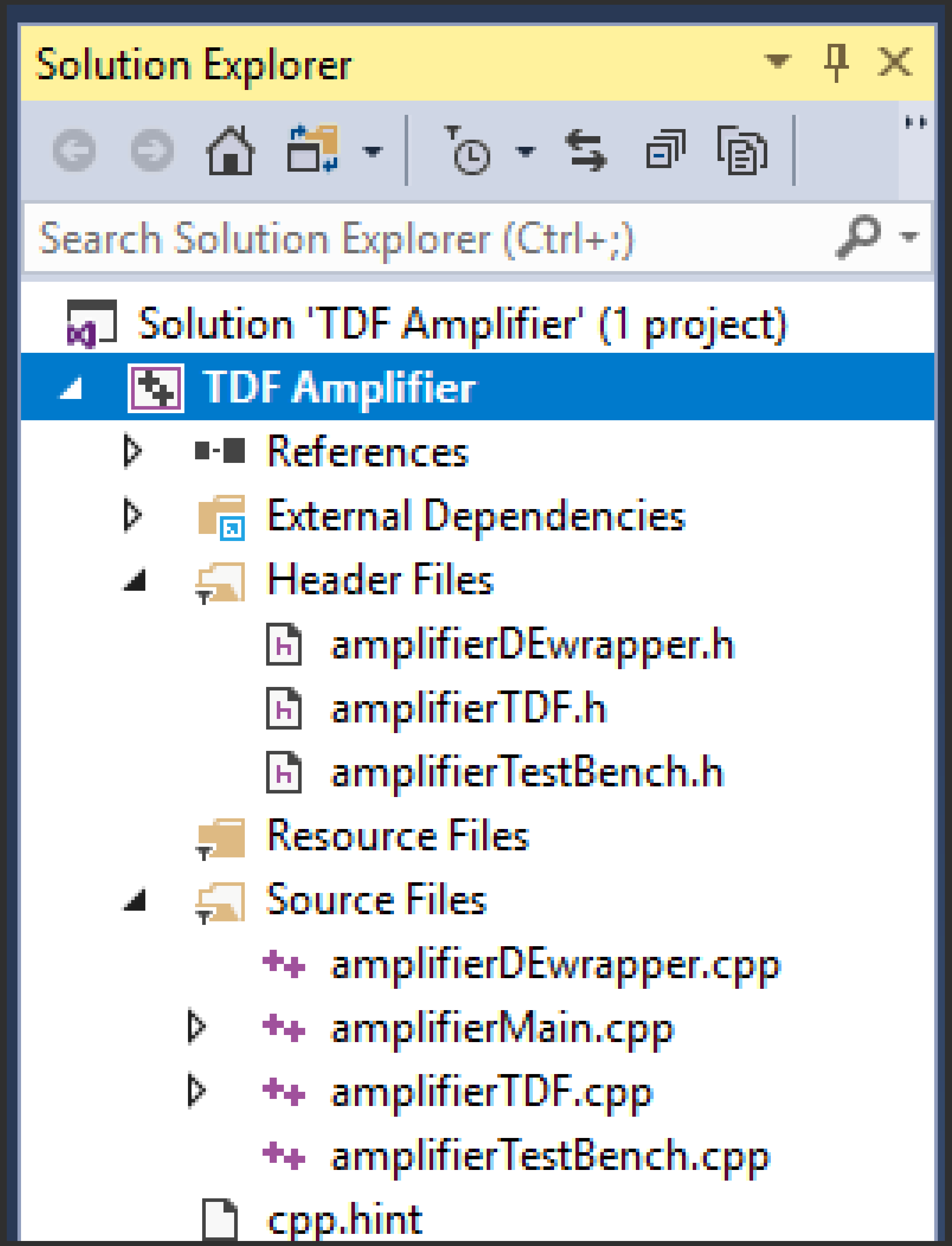
Out= sin (2*pi*f*t)

A simple TDF example – sine wave

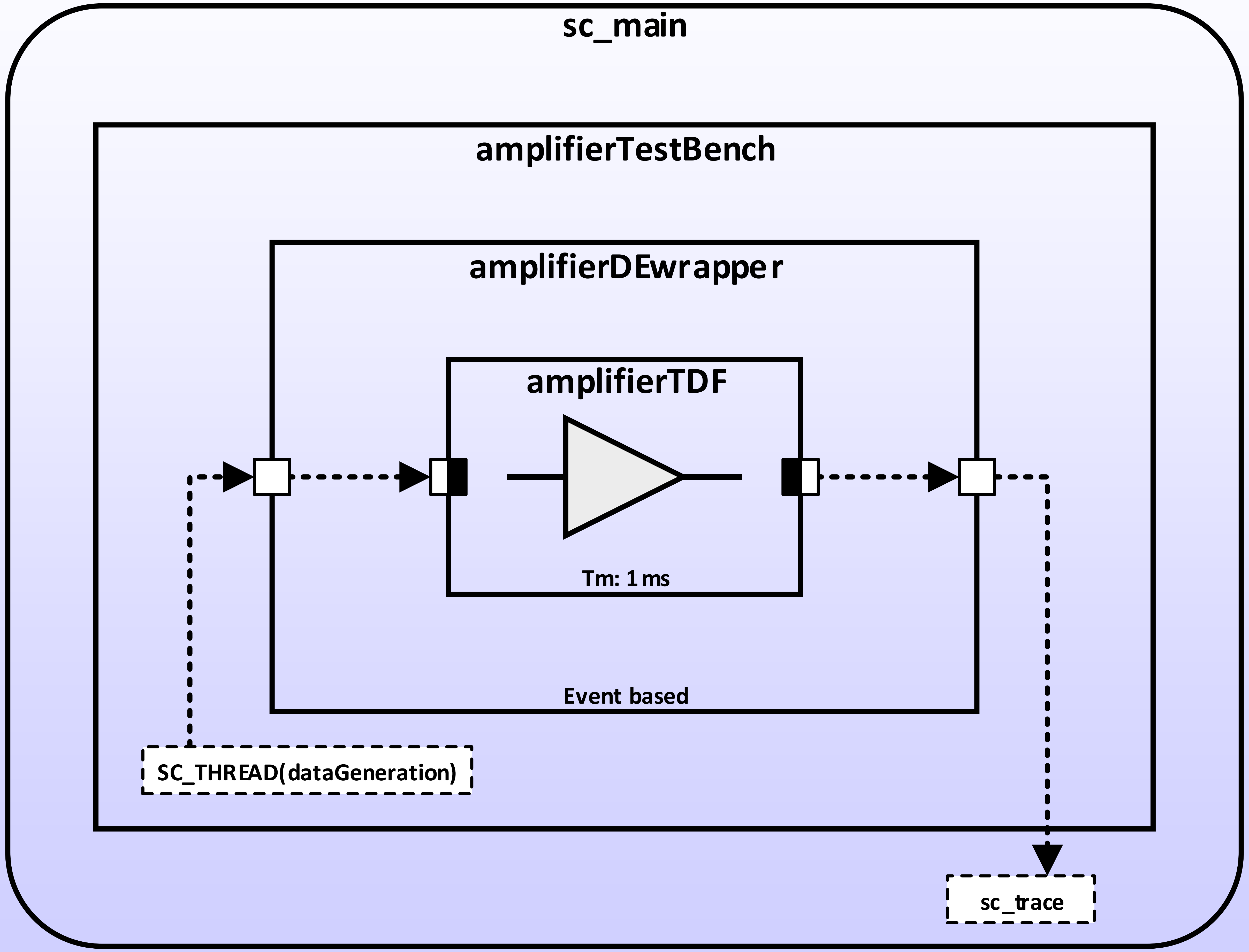
```cpp
#include "systemc-ams.h"
int sc_main(int argc, char* argv[]) {
    sc_set_time_resolution(10.0, SC_NS);
    sca_tdf::sca_signal<double> sig_sine;
    sine sin("sin");
    sin.out(sig_sine);

    sca_trace_file* tr = sca_create_vcd_trace_file("tr"); // Usual SystemC tracing
    sca_trace(tr, sig_sine, "sig_sine");
    sc_start(100, SC_MS);
    return 0;
}
```



Example: Wrapped TDF Amplifier



A simple hierarchical TDF example - amplifier



A simple hierarchical TDF example - amplifier

amplifierMain.cpp | amplifierTDF.h | amplifierDEwrapper.h | amplifierTDF.cpp

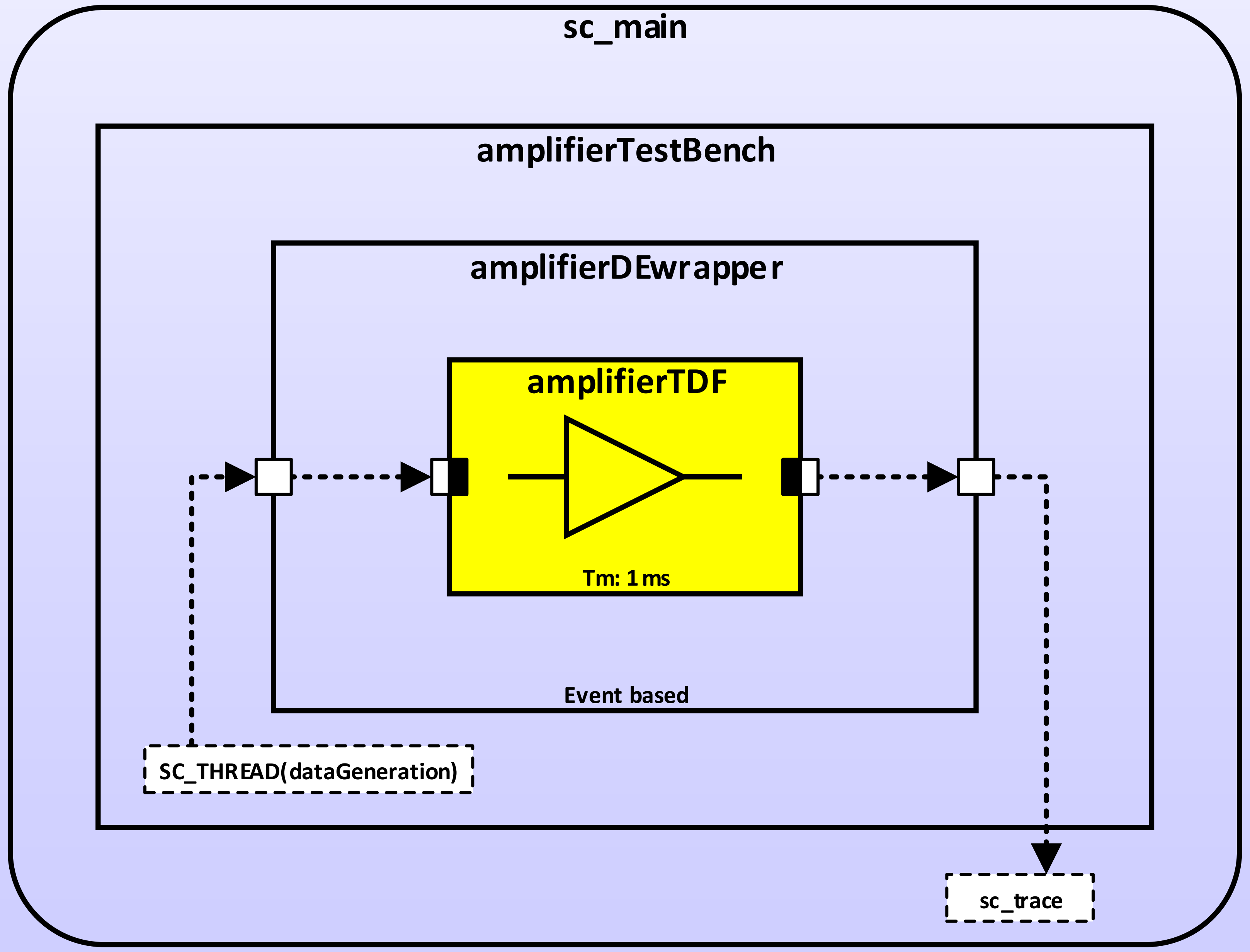
TDF Amplifier | (Global Scope)

```cpp
#include <systemc.h>
#include <systemc-ams.h>

SCA_TDF_MODULE(amplifierTDF){
    sca_tdf::sca_de::sca_in<double> inTDF;
    sca_tdf::sca_de::sca_out<double> outTDF;

    SCA_CTOR(amplifierTDF) : inTDF("inTDF"), outTDF("outTDF") {}

    void set_attributes();
    void processing();
};
```

amplifierMain.cpp | amplifierTDF.h | amplifierDEwrapper.h | amplifierTDF.cpp

TDF Amplifier | (Global Scope)

```cpp
#include "amplifierTDF.h"

void amplifierTDF::set_attributes()
{
    set_timestep(1.0, SC_MS);
}

void amplifierTDF::processing()
{
    outTDF.write(inTDF.read() * 2.0);
}
```

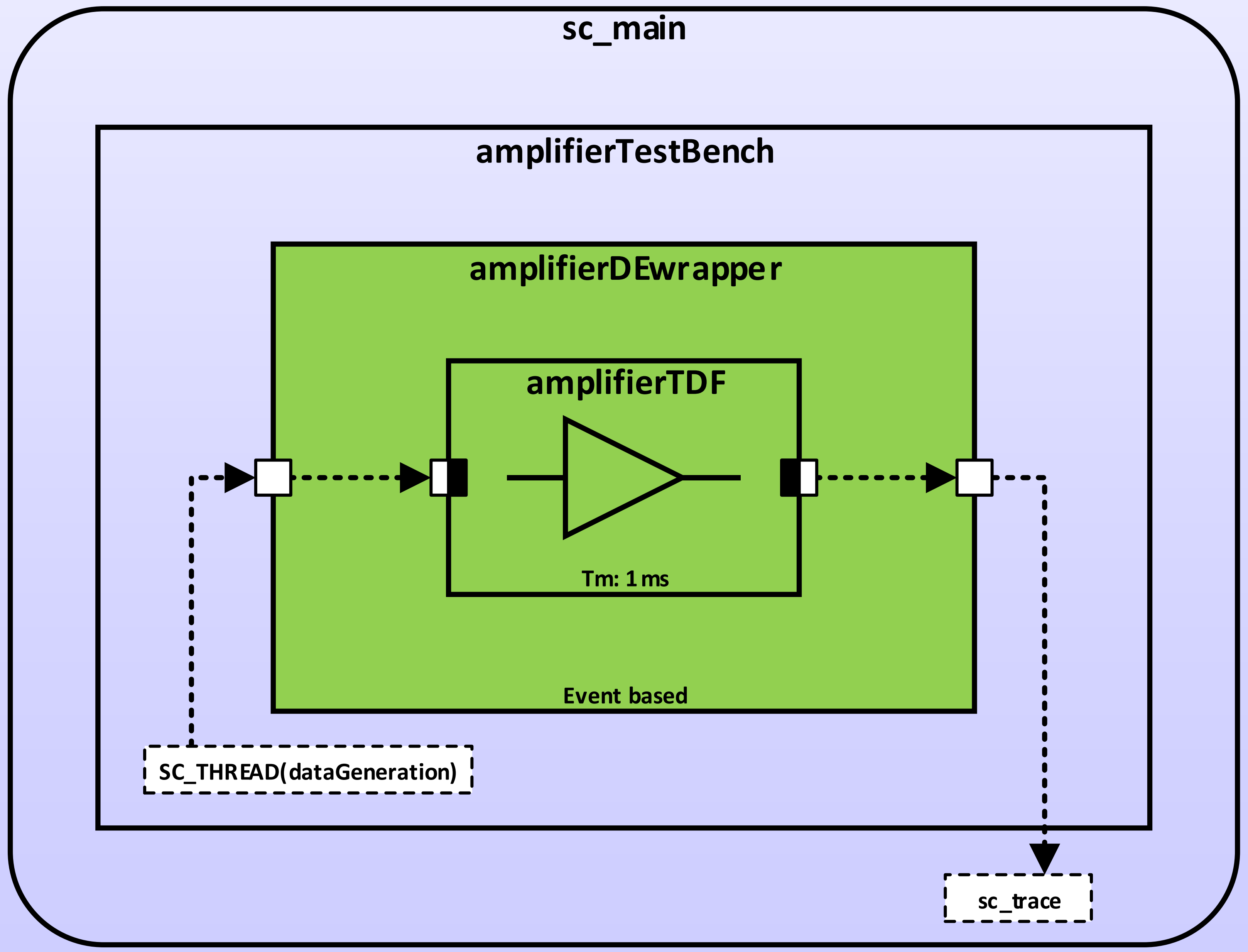


A simple hierarchical TDF example - amplifier

amplifierMain.cpp amplifierTDF.h amplifierDEwrapper.h amplifierTDF.cpp

TDF Amplifier (Global Scope)

```
#include "amplifierTDF.h"

SC_MODULE(amplifierDEwrapper){
    sc_in  <double> inDE;
    sc_out  <double> outDE;

    // primitive TDF module instantiation
    amplifierTDF* AMP1;

    SC_CTOR(amplifierDEwrapper) : inDE("inDE"), outDE("outDE")
    {
        AMP1 = new amplifierTDF("amplifierTDF");
        AMP1->inTDF(inDE);
        AMP1->outTDF(outDE);
    }
};
```

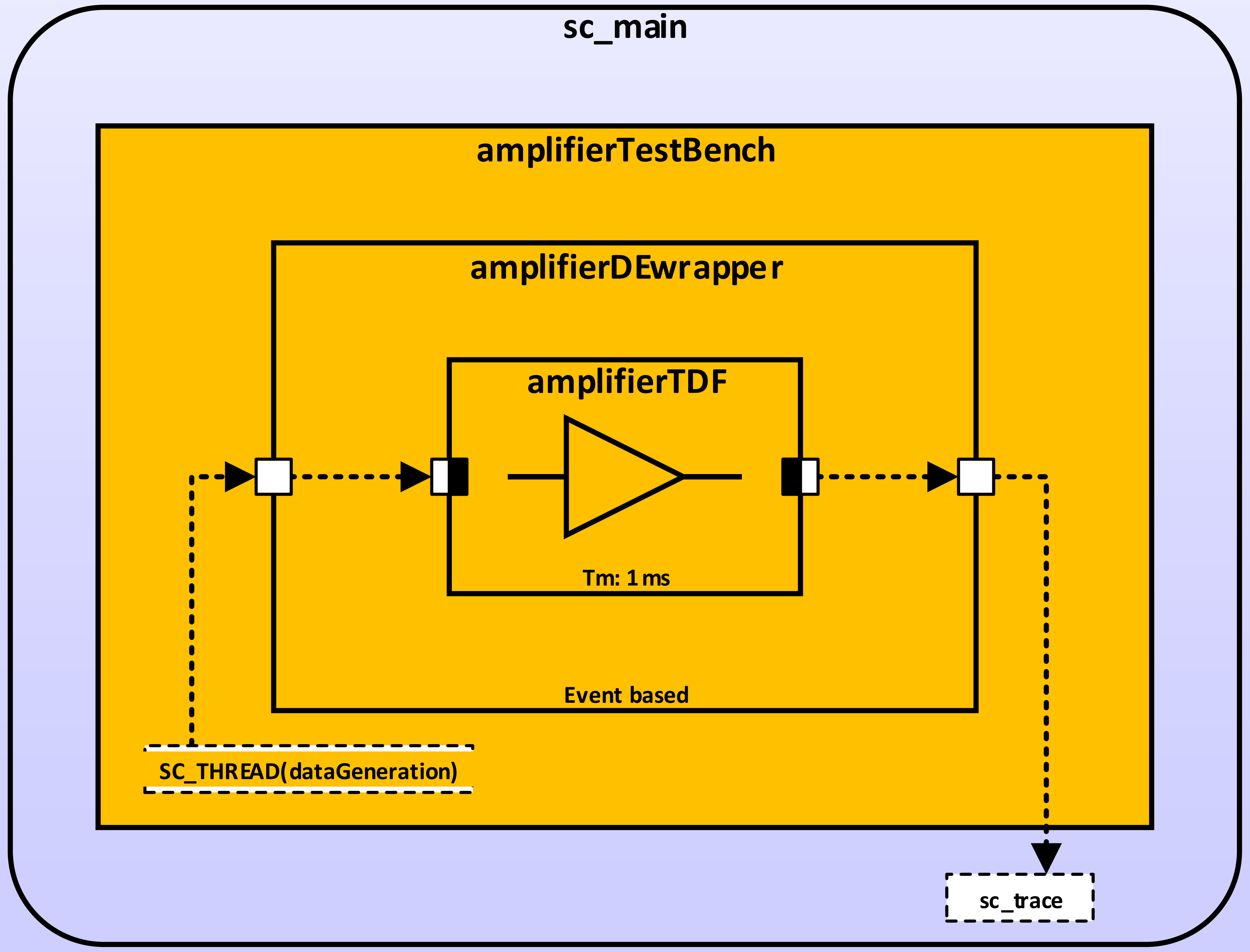


A simple hierarchical TDF example - amplifier

amplifierTDF.h | amplifierDEwrapper.cpp | amplifierTestBench.h | amplifierTDF.cpp

TDF Amplifier | (Global Scope)

```
#include "amplifierDEwrapper.h"

SC_MODULE(amplifierTestBench)
{
    sc_signal<double> ain;
    sc_signal<double> aout;

    amplifierDEwrapper* UUT;
    SC_CTOR(amplifierTestBench)
    {
        UUT = new amplifierDEwrapper("amplifierDE_instance");
            UUT->inDE(ain);
            UUT->outDE(aout);
        SC_THREAD(dataGeneration);
    }
    void dataGeneration()
    {
        ain = 23.0;
        while (true)
        {
            wait(1.4, SC_MS);
            ain = ain + 15.5;
            wait(1.5, SC_MS);
            ain = ain + 19.1;
            wait(1.1, SC_MS);
            ain = ain + 23.3;
            wait(1.4, SC_MS);
            ain = 23.3;
        }
    }
};
```

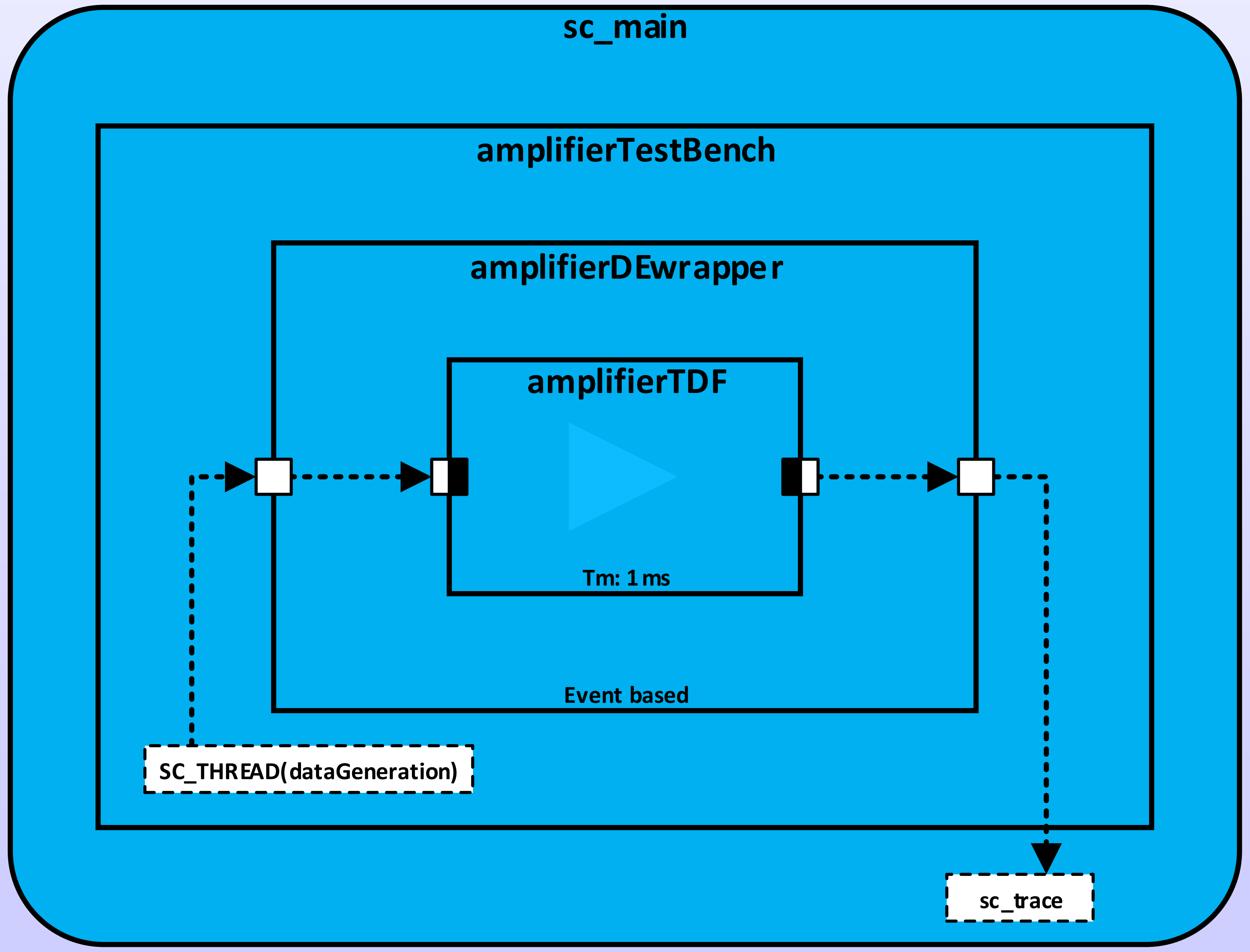


A simple hierarchical TDF example - amplifier

amplifierDEwrapper.cpp | amplifierTestBench.h | amplifierTestBench.cpp | amplifierMain.cpp

TDF Amplifier | (Global Scope)

```
#include <systemc-ams.h>
#include "amplifierTestBench.h"

int sc_main(int argc, char* argv[]){

    sc_signal <double> ain, aout;

    amplifierTestBench ATB ("amplifierTestBench");

    sc_trace_file* traceFile = sc_create_vcd_trace_file("traceFile");
    sca_trace_file* traceFileAMS = sca_create_tabular_trace_file( "traceFileAMS.dat" );

    sc_trace(traceFile, ATB.ain, "aInput");
    sc_trace(traceFile, ATB.aout, "aOutput");

    sca_trace(traceFileAMS, ATB.ain, "aInput");
    sca_trace(traceFileAMS, ATB.aout, "aOutput");

    sc_start(1000, SC_MS);
    return 0;
}
```

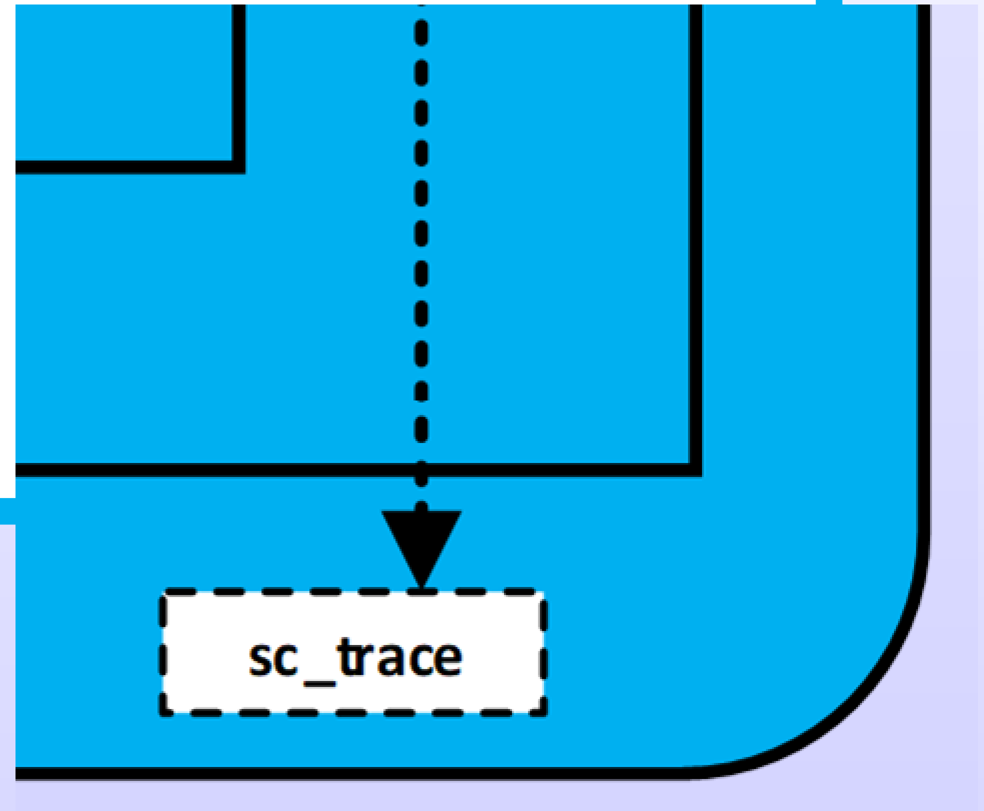


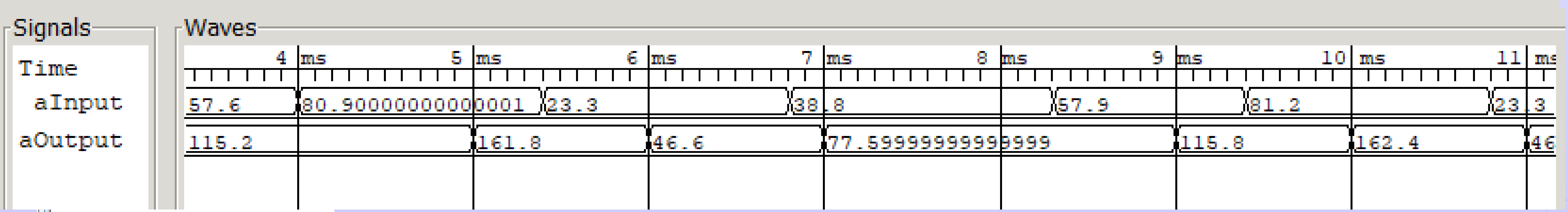
A simple hierarchical TDF example - amplifier

traceFileAMS.dat

```
%time aInput aOutput
0 23 0
0.001 23 46
0.0014 38.5 46
0.002 38.5 77
0.0029 57.6 77
0.003 57.6 115.2
0.004 80.9 115.2
0.005 80.9 161.8
0.0054 23.3 161.8
0.006 23.3 46.6
0.0068 38.8 46.6
0.007 38.8 77.6
0.0083 57.9 77.6
0.009 57.9 115.8
0.0094 81.2 115.8
0.01 81.2 162.4
0.0108 23.3 162.4
0.011 23.3 46.6
0.0122 38.8 46.6
0.013 38.8 77.6
0.0137 57.9 77.6
0.014 57.9 115.8
0.0148 81.2 115.8
0.015 81.2 162.4
0.0162 23.3 162.4
0.017 23.3 46.6
0.0176 38.8 46.6
...
0.0256 81.2 115.8
```

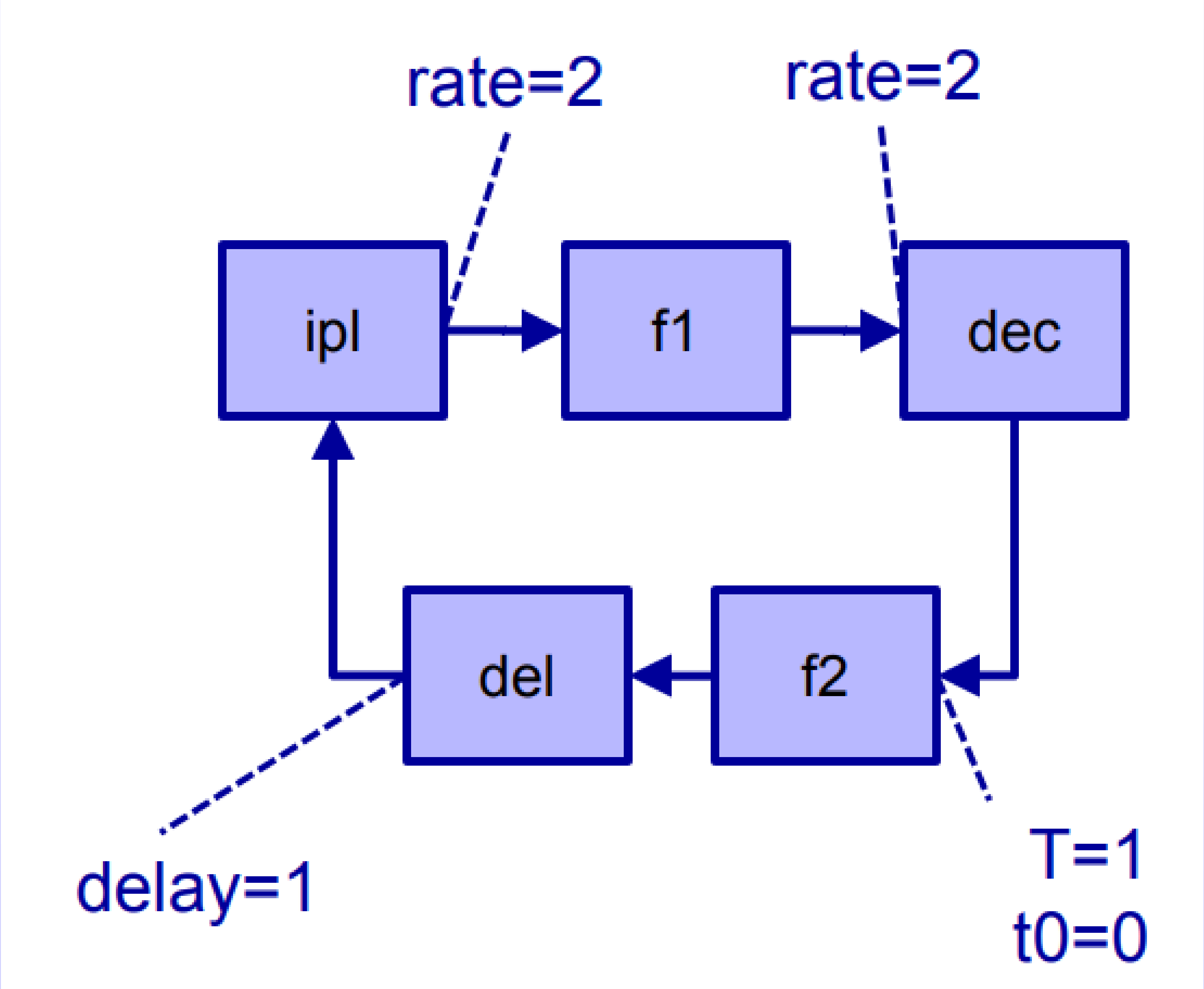
```
sc_trace_file* traceFile = sc_create_vcd_trace_file("traceFile");
sca_trace_file* traceFileAMS = sca_create_tabular_trace_file( "traceFileAMS.dat" );

sc_trace(traceFile, ATB.ain, "aInput");
sc_trace(traceFile, ATB.aout, "aOutput");

sca_trace(traceFileAMS, ATB.ain, "aInput");
sca_trace(traceFileAMS, ATB.aout, "aOutput");
```

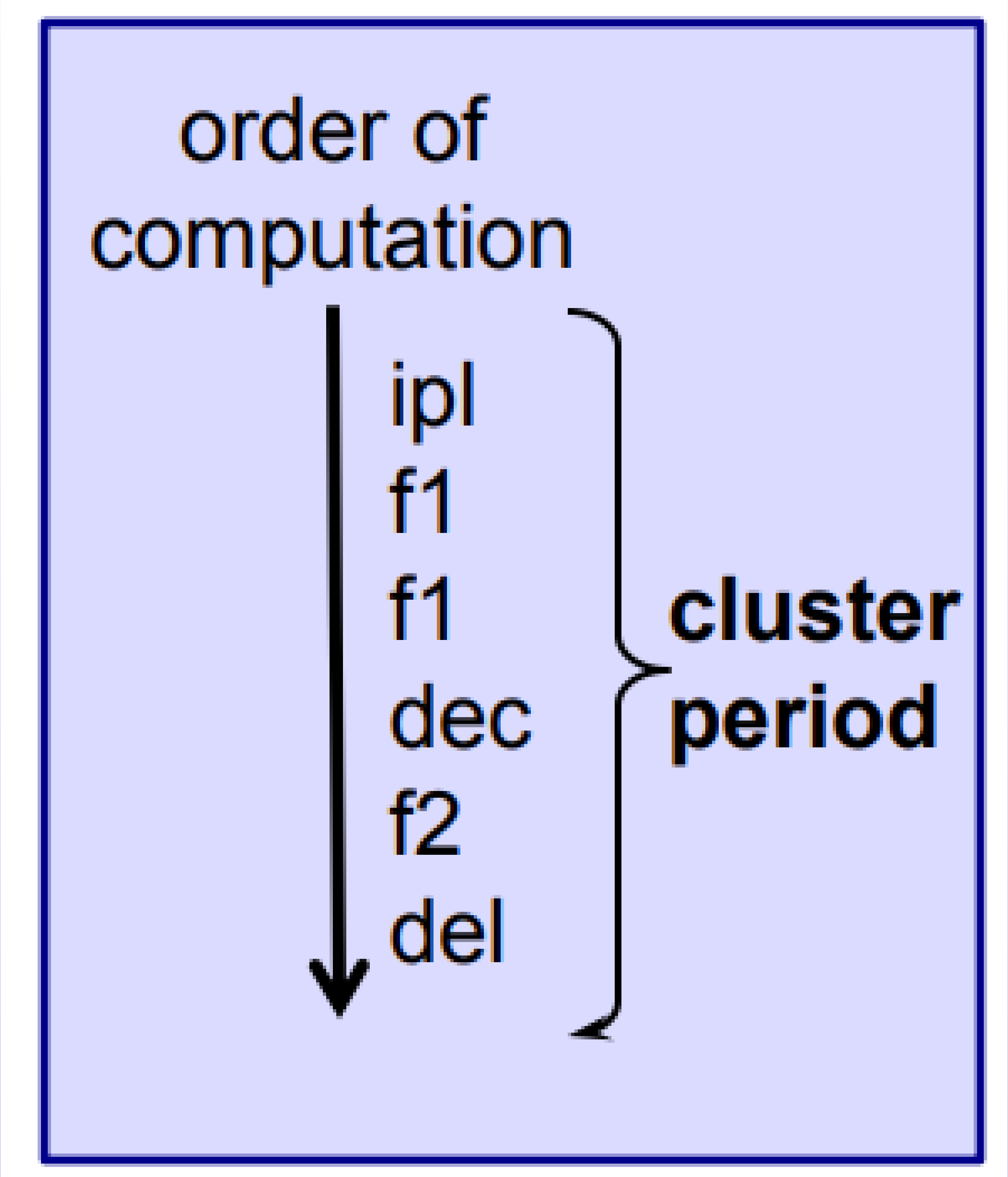




TDF Module and Port Attributes

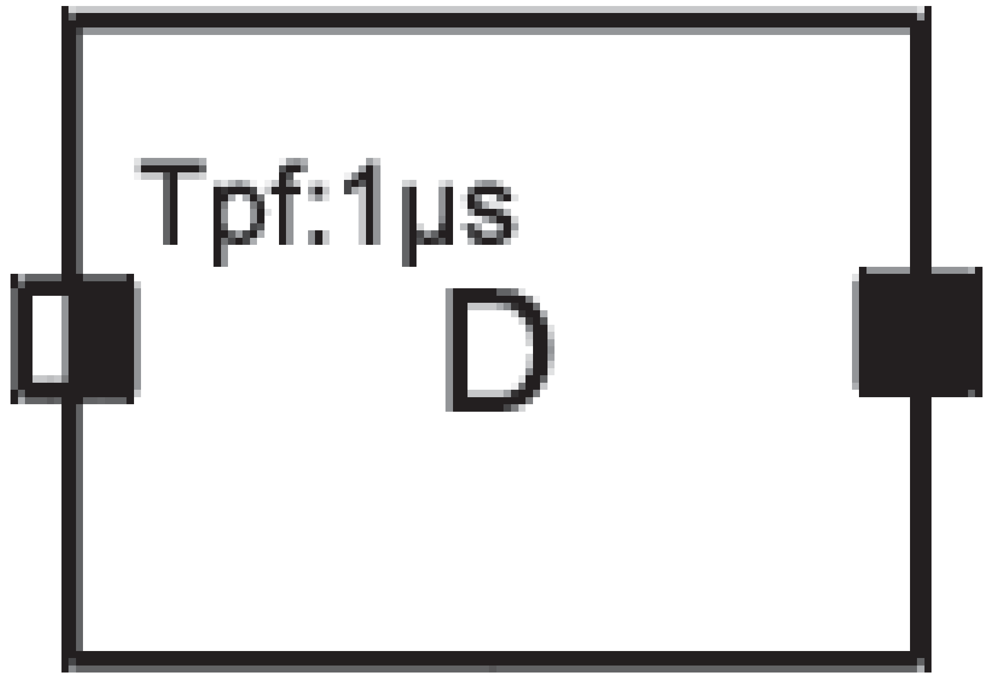
„cluster“ := set of connected TDF modules





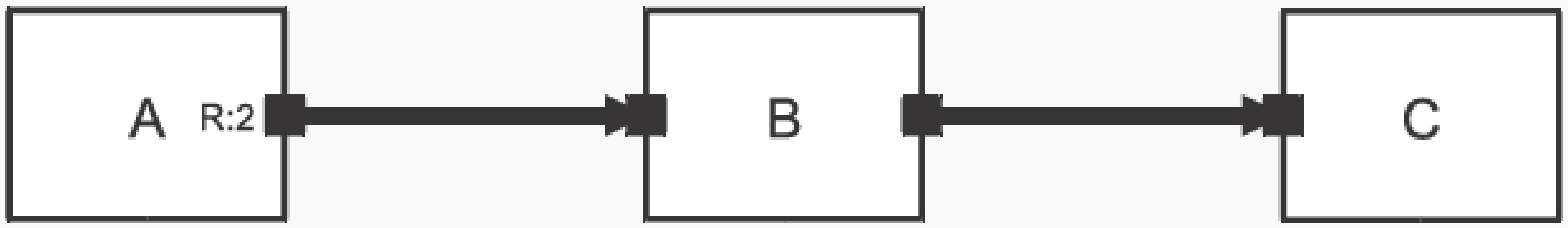
TDF Module and Port Attributes

- The main advantage: Execution of TDF models does not rely on the evaluate/ update mechanism of SystemC's discrete-event kernel , therefore, can be simulated more efficiently



Port time offset

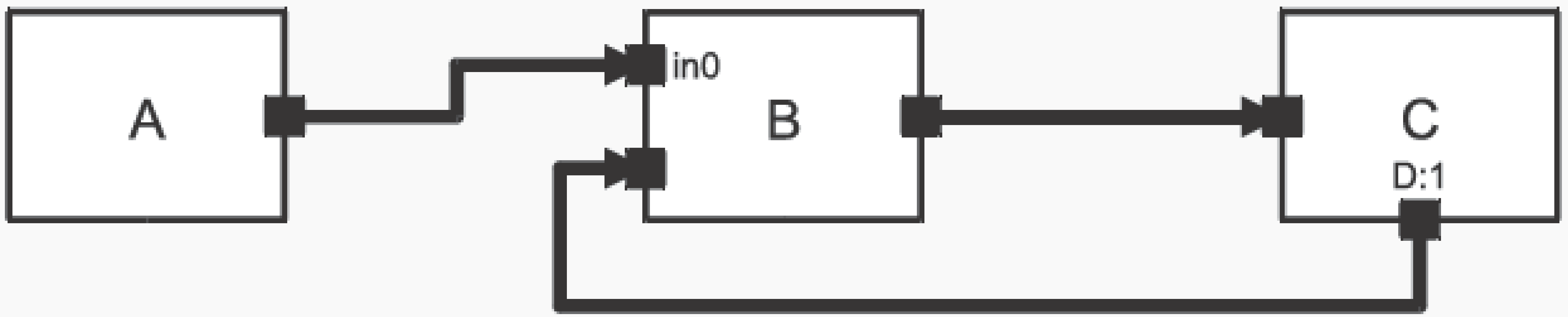
TDF Model Topologies- Scheduling



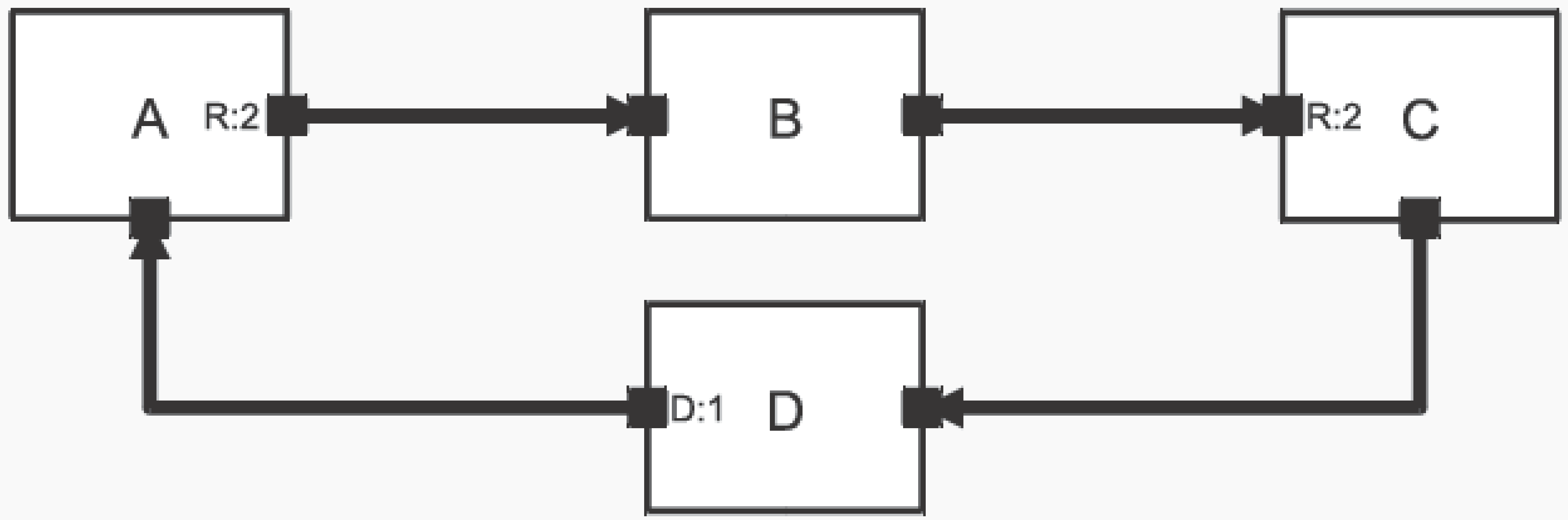
Possible schedule: {A→B→C→B→C}

Possible schedule: {B→A}

TDF Model Topologies- Scheduling



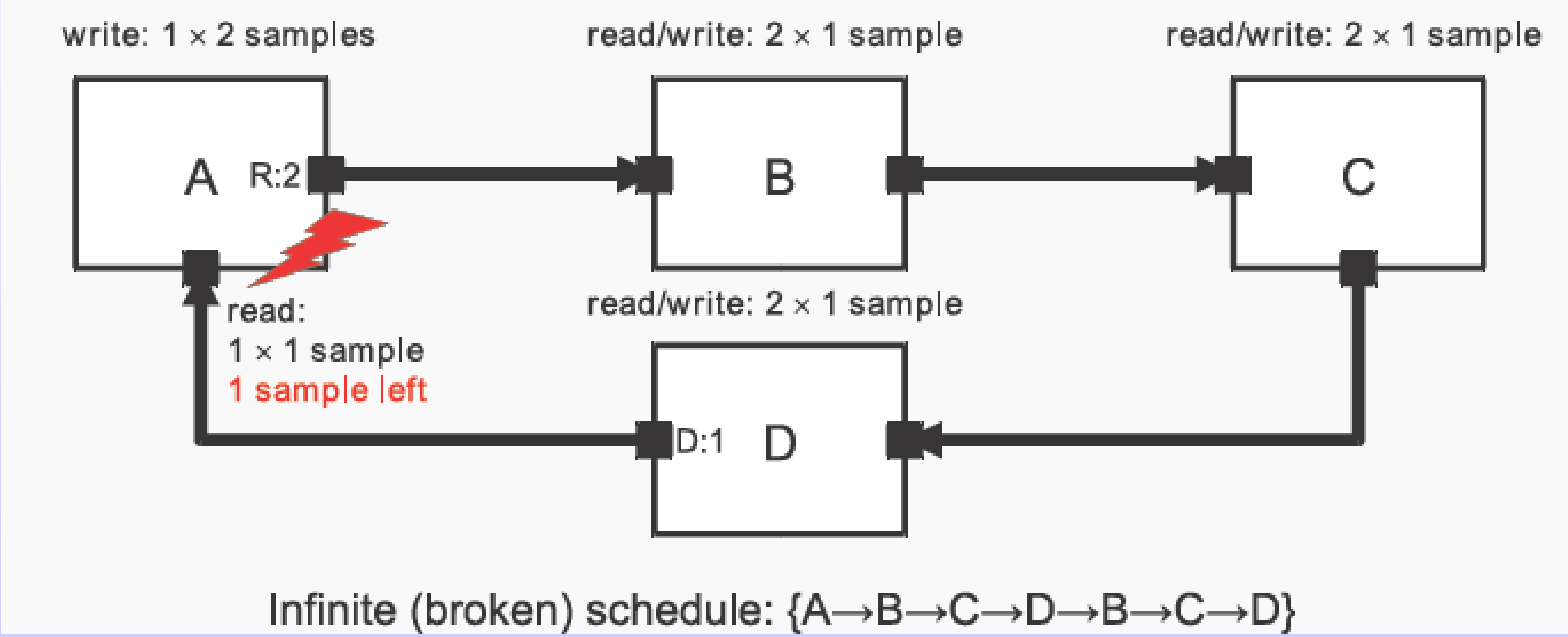
Possible schedule: {A→B→C}



Possible schedule: {A→B→B→C→D}

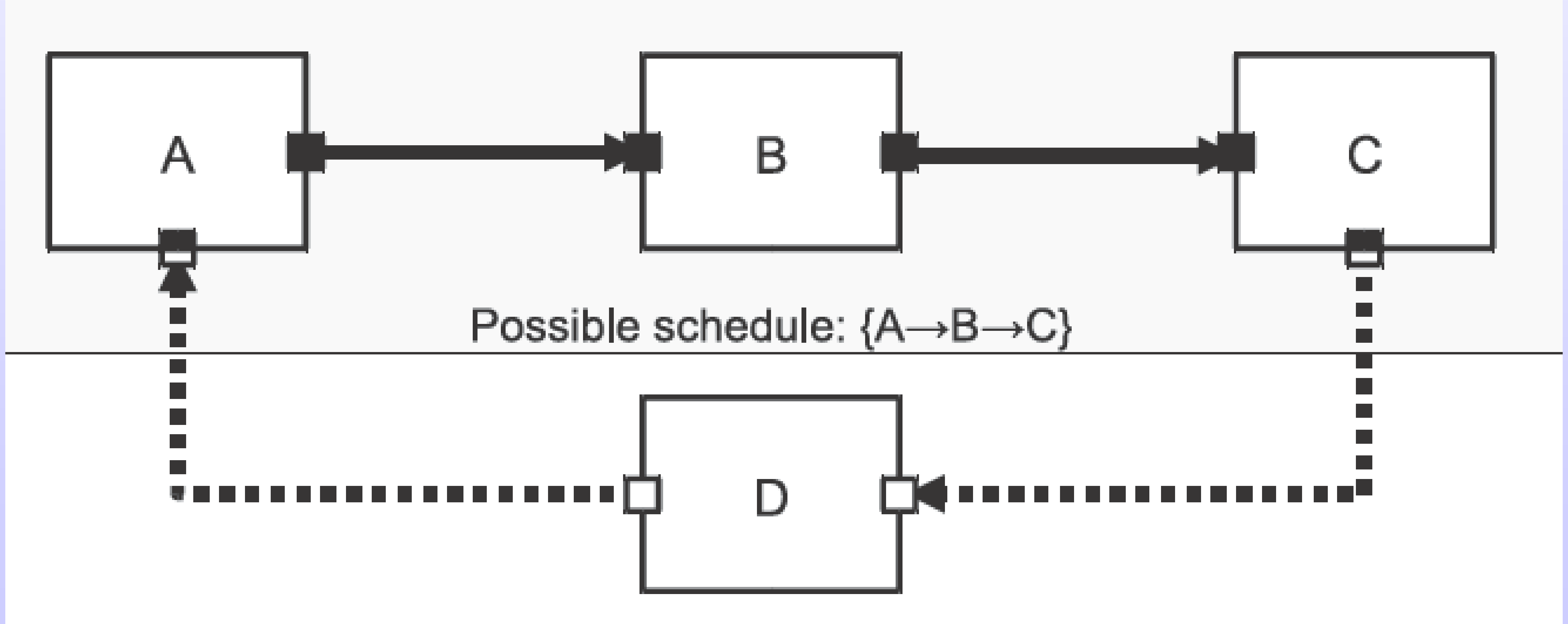
TDF Model Topologies- Scheduling

- Prerequisite for a proper schedule
  - The sum of samples produced at the output ports within a loop must be equal to the sum of samples consumed by the input ports within the loop

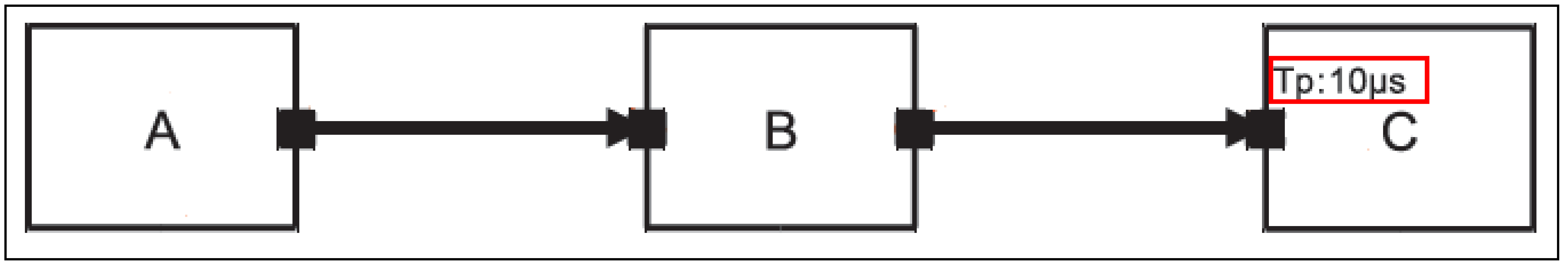


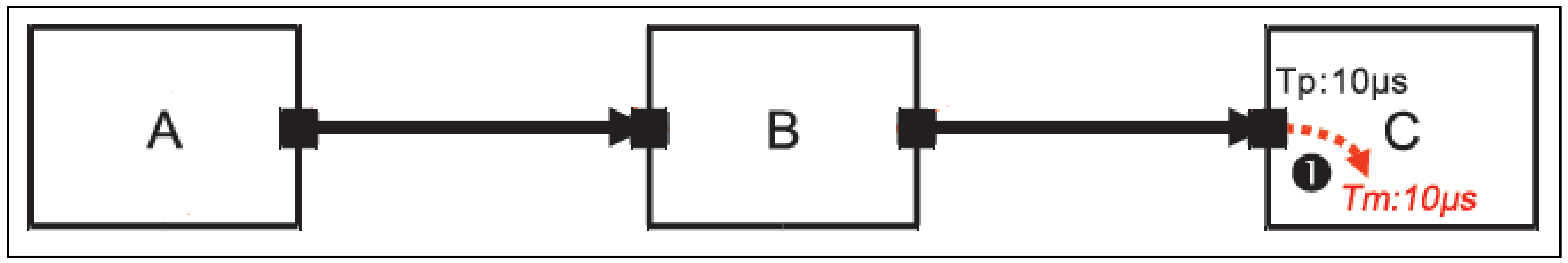
TDF Model Topologies- Scheduling

- Note :
  - TDF samples read from module C and passed through the discrete-event module D to the input of module A will be delayed by one TDF time step due to the evaluate/update mechanism of the SystemC kernel.

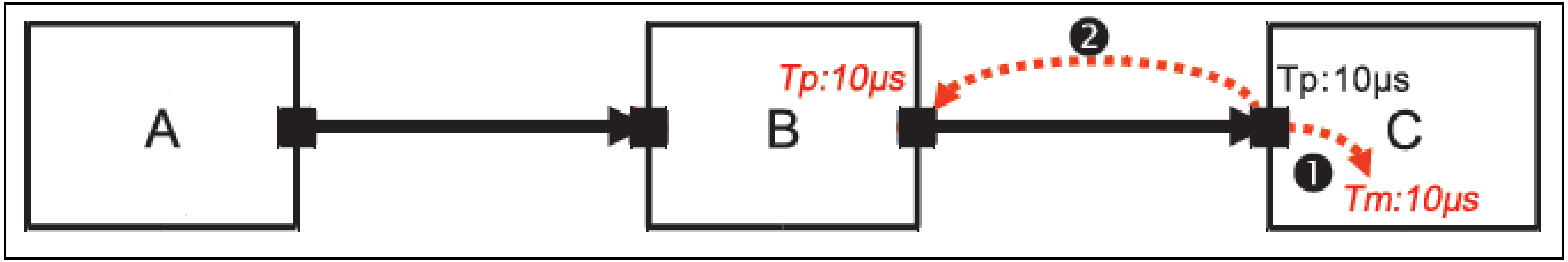


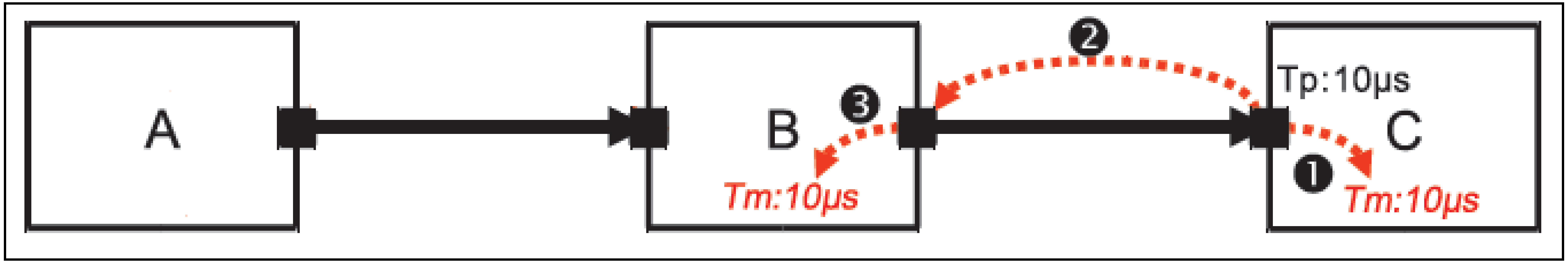
Time Step Assignment and Propagation



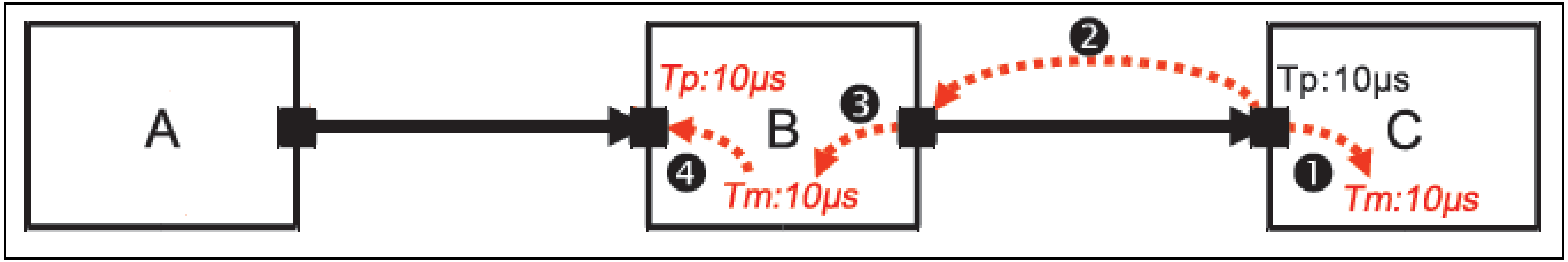


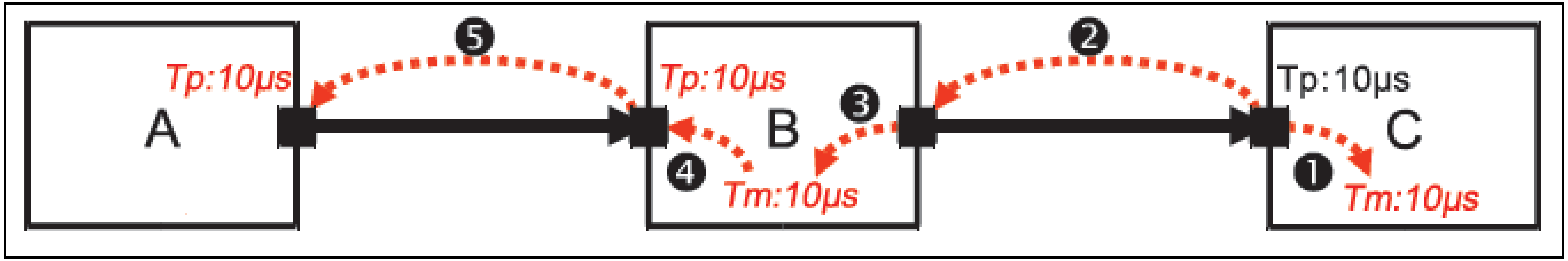
Time Step Assignment and Propagation



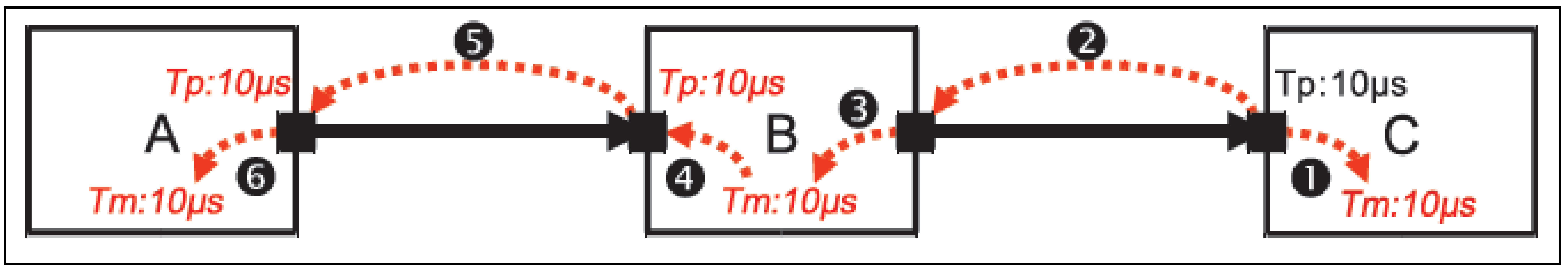


Time Step Assignment and Propagation





Time Step Assignment and Propagation



Time Step Assignment and Propagation

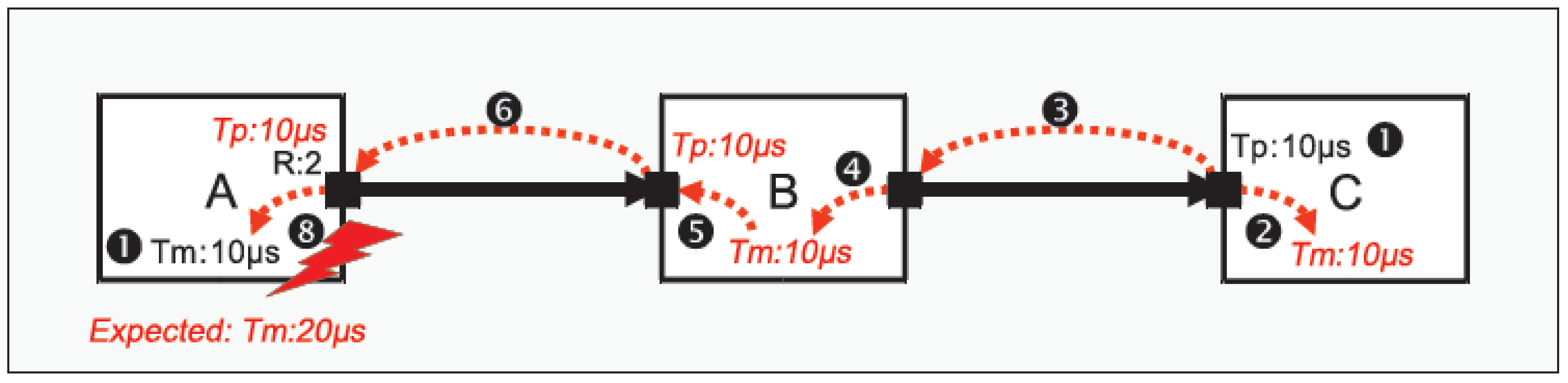
Consistency of time step assignment and propagation :

Module time-step (Tm)

= Input port time-step (Tpi) . Input port rate (Ri)

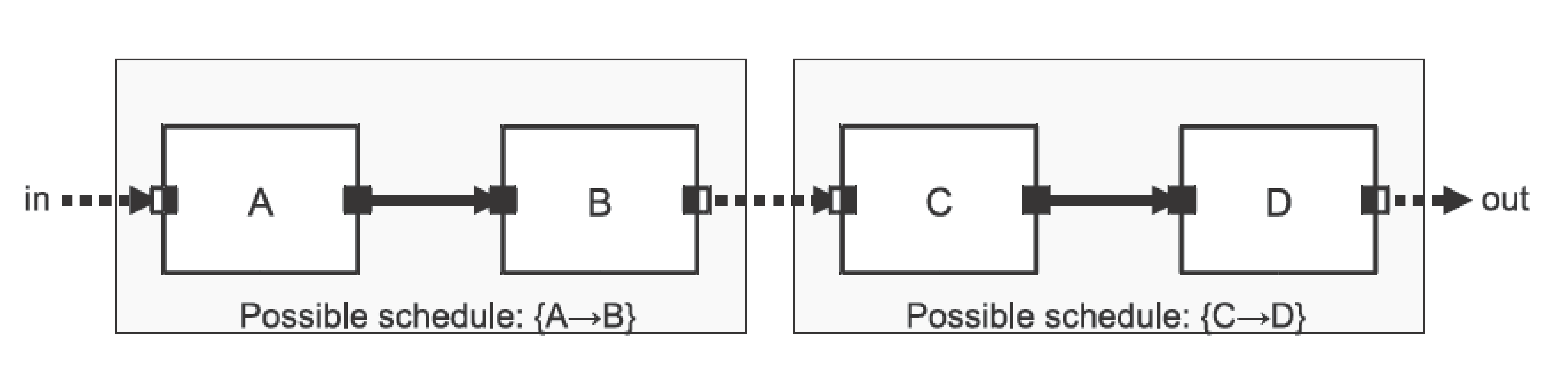
= Output port time-step (Tpo) . Output port rate (Ro)

Time Step Assignment and Propagation



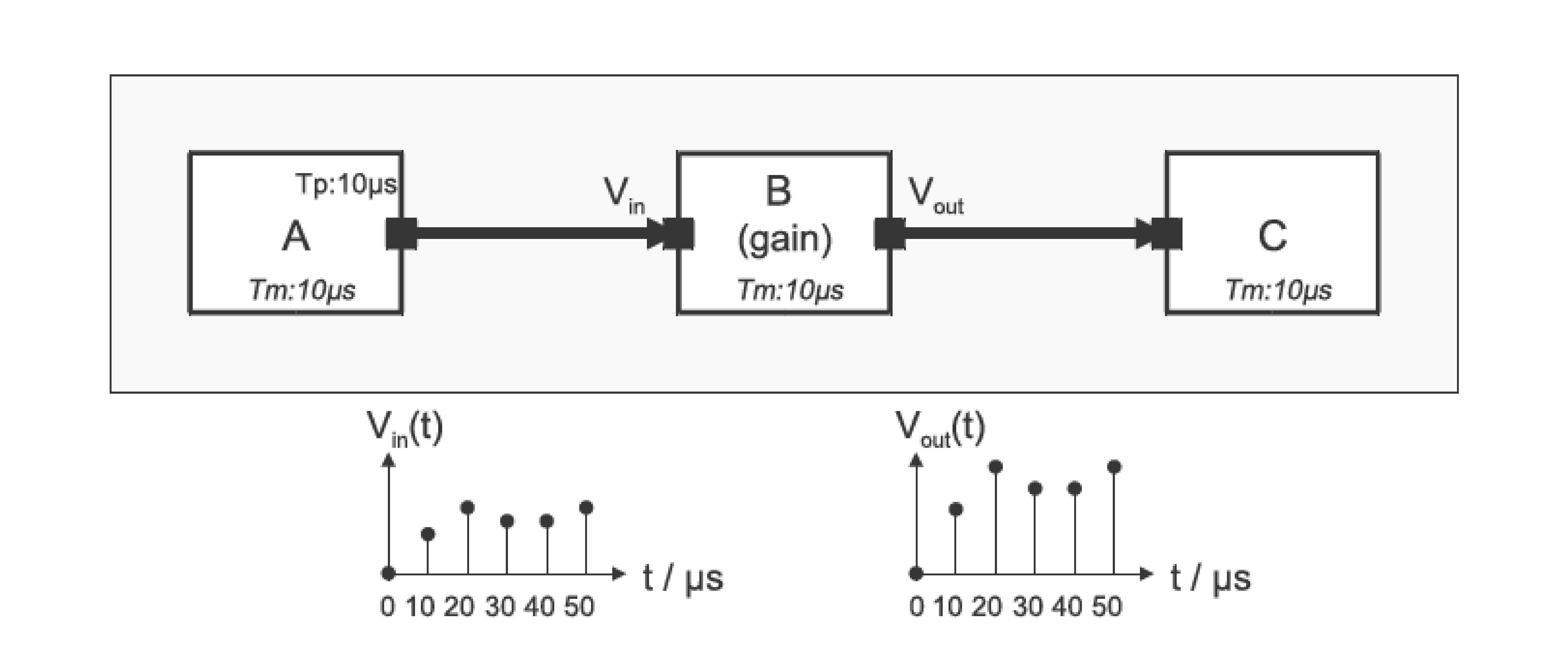
Multiple Schedule or Cluster

- It is possible to have more than one TDF cluster within the same application.
- Each TDF cluster has its own data flow characteristics (sampling rate, sampling period, etc), scheduling and execution order
- The main element to indirectly change the cluster structure, is to use the TDF converter ports



Signal Processing Behavior of TDF Models

- A cluster of TDF modules processes signals by repetitively activating the processing functions of the contained modules in the order of the derived schedule
- Samples are generated for each module as a function of time.

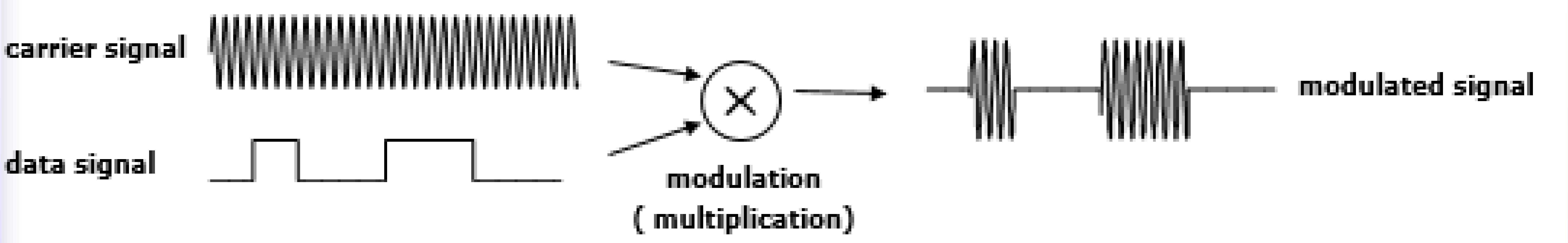


A hierarchy TDF example

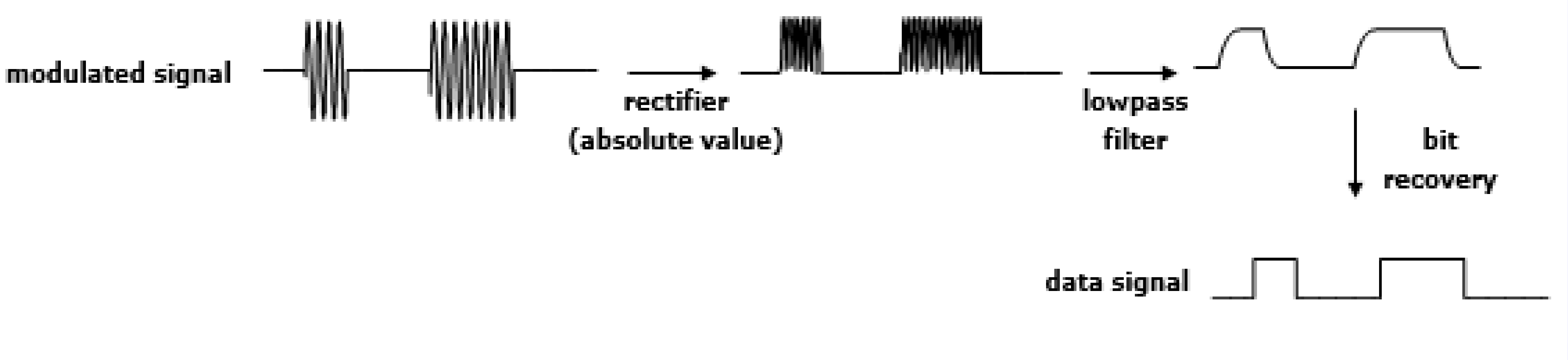
A BASK modulator-demodulator

•BASK: Binary Amplitude Shift keying

• Principle of BASK modulation:

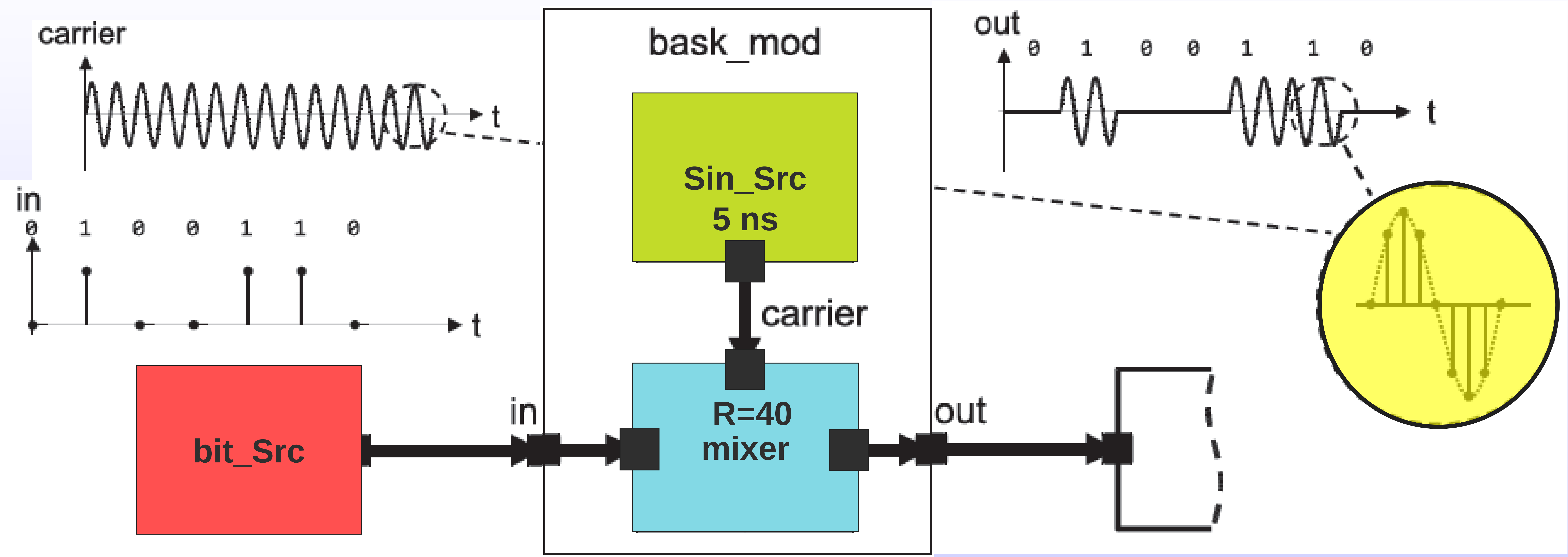


• Principle of BASK de-modulation:



A hierarchy TDF example

A BASK modulator



A hierarchy TDF example

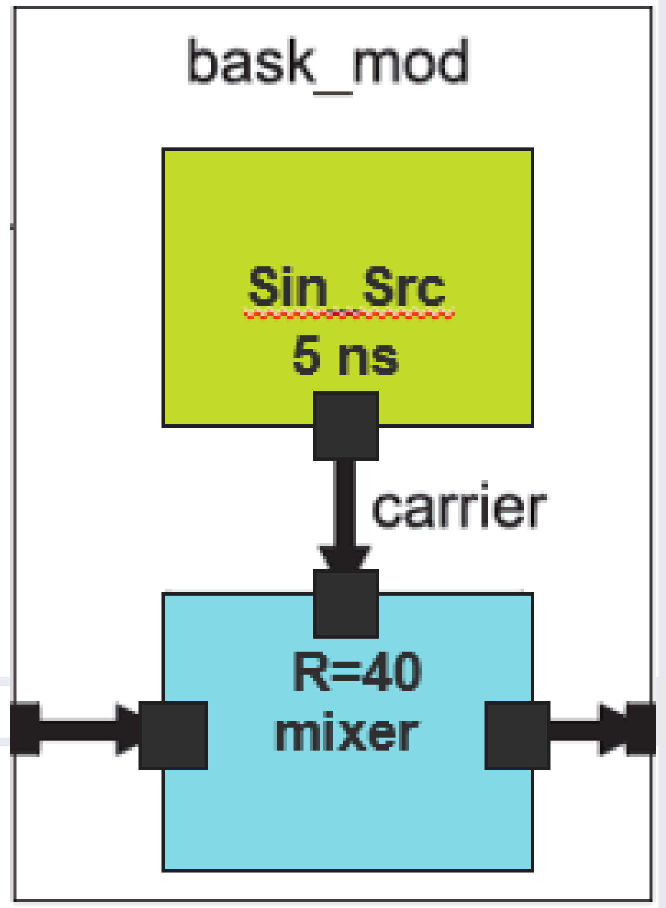
Sine source

bask_demod.h | sine_wave.h | sampler.h | mixer.h | ltf_nd_filter.h | bask_mod.h | sim.cpp

BASK | (Global Scope)

```
#include <fstream>
using namespace std;

SCA_TDF_MODULE(sin_src)
{
    sca_tdf::sca_out<double> out; // output port

    sin_src( sc_core::sc_module_name nm, double ampl_= 1.0, double freq_ = 1.0e3,
             sca_core::sca_time Tm_ = sca_core::sca_time(0.125, sc_core::SC_MS) )
             : out("out"), ampl(ampl_), freq(freq_), Tm(Tm_) {}

    void set_attributes()
    {
        set_timestep(Tm);
    }
    void processing()
    {
        double t = get_time().to_seconds(); // actual time
        out.write( ampl * std::sin( 2.0 * PI * freq * t ) );
    }

    private:
        double ampl; // amplitude
        double freq; // frequency
        sca_core::sca_time Tm; // module time step
};
```

100 %



A hierarchy TDF example

Mixer

bask_demod.h | sine_wave.h | sampler.h | mixer.h | ltf_nd_filter.h | bask_mod.h | sim.cpp

BASK | (Global Scope)

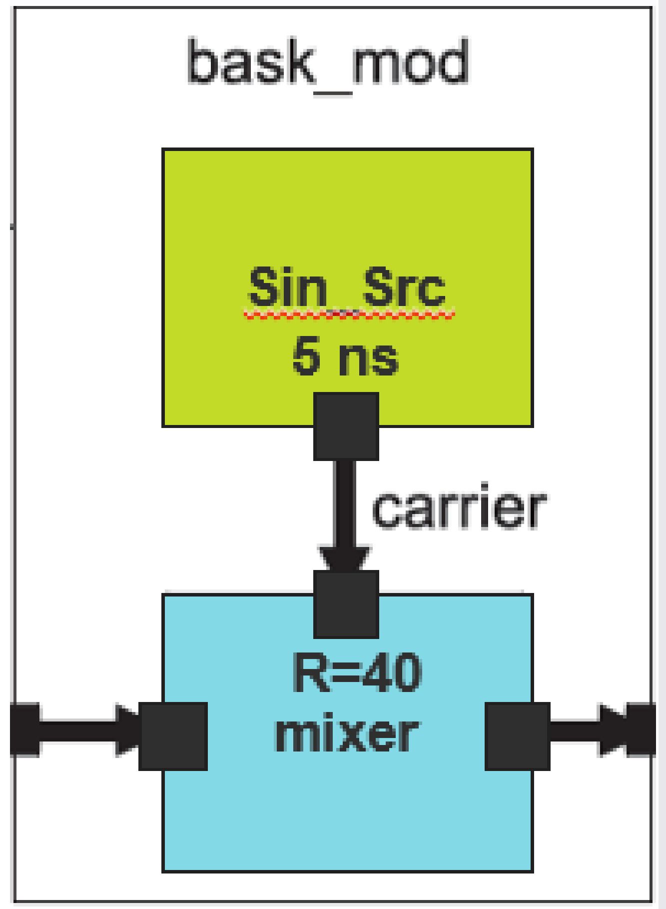
```
SCA_TDF_MODULE(mixer)
{
    sca_tdf::sca_in<bool> in_bin; // input port baseband signal
    sca_tdf::sca_in<double> in_wav; // input port carrier signal
    sca_tdf::sca_out<double> out; // output port modulated signal

    SCA_CTOR(mixer)
    : in_bin("in_bin"), in_wav("in_wav"), out("out"), rate(40) {}
    // ... carrier data ra...

            ...e);

    void processing()
    {
        for(unsigned long i = 0; i < rate; i++)
        {
            if ( in_bin.read() )
            out.write( in_wav.read(i), i );
            else
            out.write( 0.0, i );
        }
    }
private:
    unsigned long rate;
};
```

100 %

Time_step=5*40 =200ns

Time_step =5ns

Baseband_freq = 5MHz



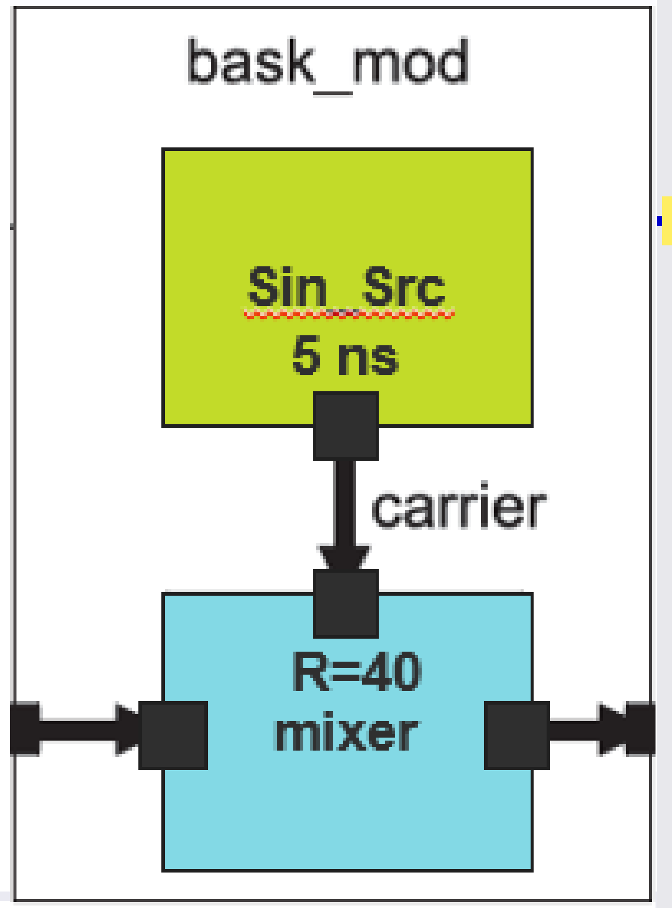
A hierarchy TDF example

BASK modulator

bask_demod.h | sine_wave.h | sampler.h | mixer.h | ltf_nd_filter.h | bask_mod.h* | sim.cpp

BASK | (Global Scope)

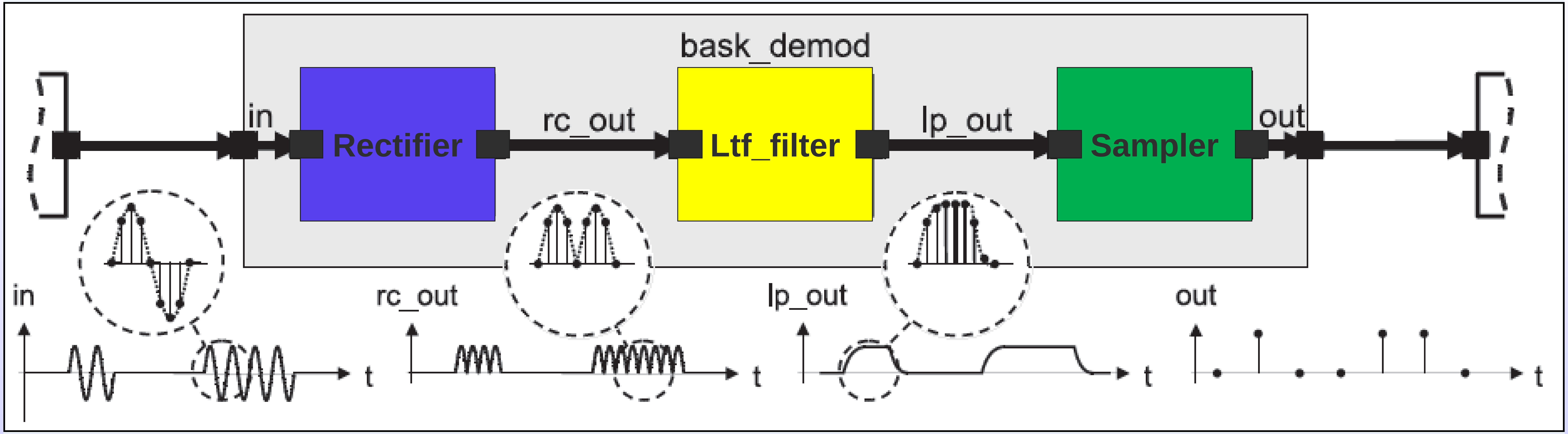
```
SC_MODULE(bask_mod)
{
    sca_tdf::sca_in<bool> in;
    sca_tdf::sca_out<double> out;
    sca_tdf::sca_out<double> carrier;

    sin_src sine;
    mixer mix;

    SC_CTOR(bask_mod)
    : in("in"), out("out"),
    sine("sine", 1.0, 1.0e7, sca_core::sca_time( 5.0, sc_core::SC_NS ) ),
    mix("mix")
    {
        sine.out(carrier);
        mix.in_wav(carrier);
        mix.in_bin(in);
        mix.out(out);
    }

};
```



100 %

A hierarchy TDF example

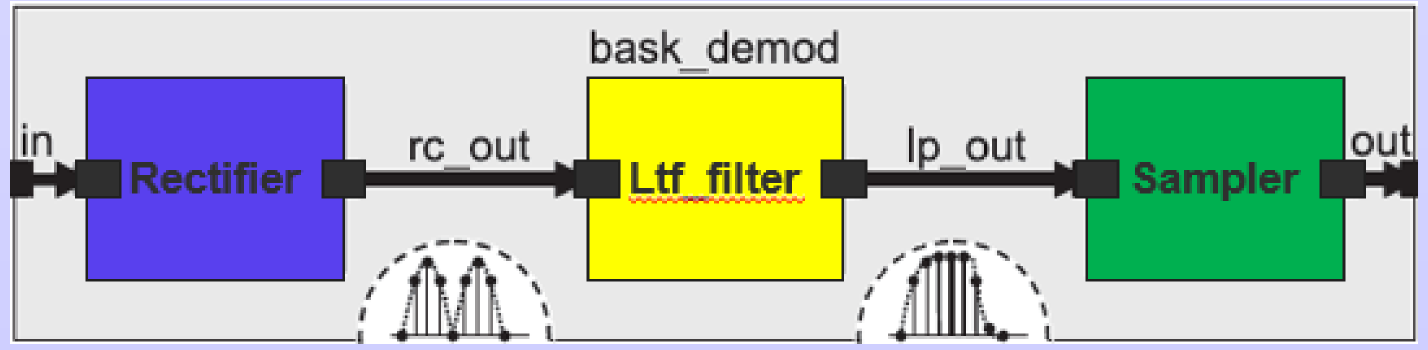
BASK Demodulator



A hierarchy TDF example

Rectifier

```
SCA_TDF_MODULE(rectifier)
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;

    SCA_CTOR(rectifier) : in("in"), out("out") {}

    void processing()
    {
        out.write( std::abs(in.read()) );
    }
};
```



A hierarchy TDF example

First-order low-pass filter using Laplace transfer function

rectifier.h | bask_demod.h | sine_wave.h | sampler.h | mixer.h | ltf_nd_filter.h | bask_mod.h | sim.cpp

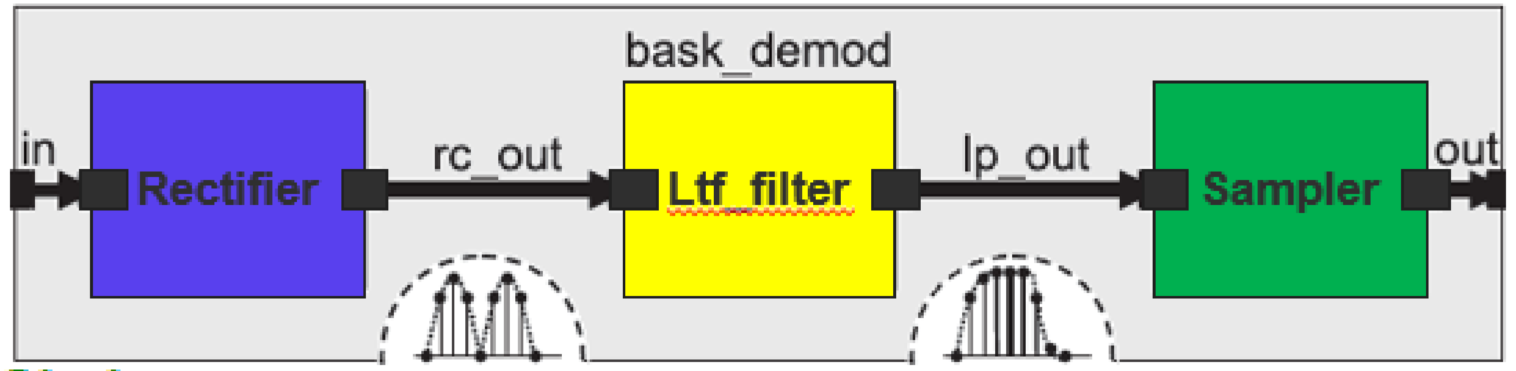
BASK | (Global Scope)

```
SCA_TDF_MODULE(ltf_nd_filter)
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;

    ltf_nd_filter( sc_core::sc_module_name nm, double fc_, double h0_ = 1.0 )
    : in("in"), out("out"), fc(fc_), h0(h0_) {}

    void initialize()
    {
        num(0) = 1.0;
        den(0) = 1.0;
        den(1) = 1.0 /( 2.0 * PI * fc );
    }
    void processing()
    {
        out.write( ltf_nd( num, den, in.read(), h0 ) );
    }
private:
        sca_tdf::sca_ltf_nd ltf_nd; // Laplace transfer function
        sca_util::sca_vector<double> num, den; // numerator and denominator coefficients
        double fc; // 3dB cut-off frequency in Hz
        double h0; // DC gain
};
```

100 %

contains 40 samples per 200 ns, and needs to get sampled down to 1 sample per 200 ns

$$H(s) = \frac{H_0}{1 + \frac{1}{2\pi f_c} s}$$



A hierarchy TDF example

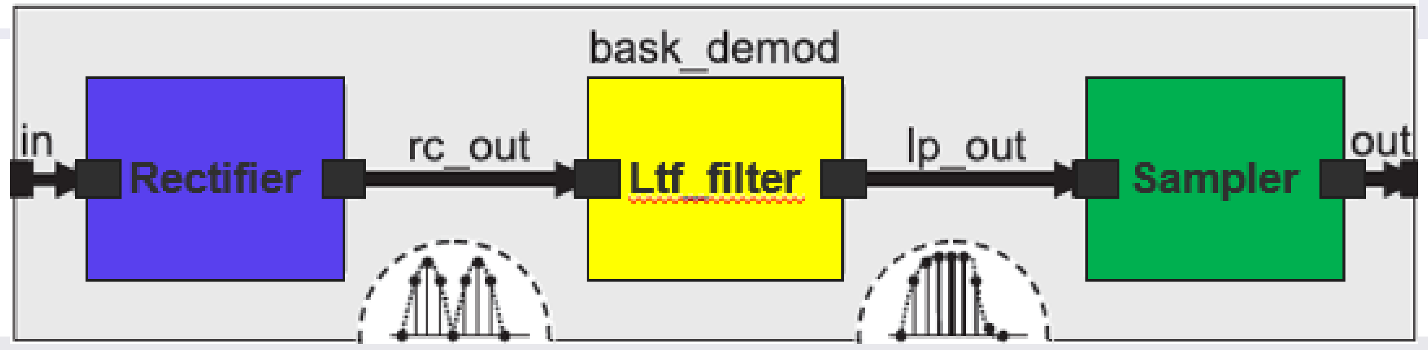
Sampler

rectifier.h | bask_demod.h | sine_wave.h | sampler.h | mixer.h | ltf_nd_filter.h | bask_mod.h | sim.cpp

BASK | (Global Scope)

```
SCA_TDF_MODULE(sampler)
{
    sca_tdf::sca_in<double> in; // input port
    sca_tdf::sca_out<bool> out; // output port

    SCA_CTOR(sampler) : in("in"), out("out"), rate(40), threshold(0.2) {}

    void set_attributes()
    {
        in.set_rate(rate);
        sample_pos = (unsigned long)std::ceil( 2.0 * (double)rate/3.0 );
    }
    void processing()
    {
        if( in.read(sample_pos) > threshold )
        out.write(true);
        else
        out.write(false);
    }
    private:
        unsigned long rate;
        double threshold;
        unsigned long sample_pos;
};
```

the output of the low-pass filter can be expected to be settled by that time.



100 %

A hierarchy TDF example

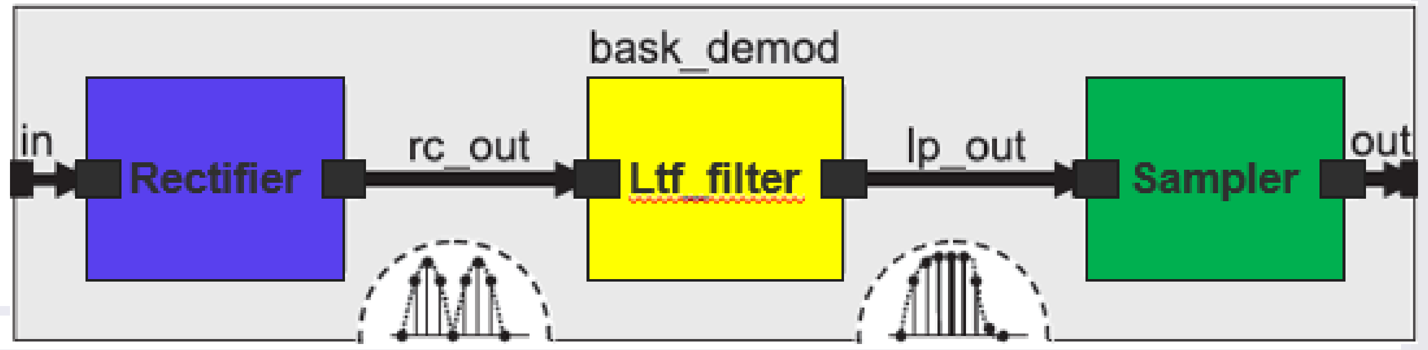
Bask Demodulator

bit_src.h | rectifier.h | bask_demod.h | sine_wave.h | sampler.h | mixer.h | ltf_nd_filter.h | bask_mod.h | sim.cpp

BASK | (Global Scope)

```
SC_MODULE(bask_demod)
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<bool> out;

    rectifier rc;
    ltf_nd_filter lp;
    sampler sp;

    SC_CTOR(bask_demod)
    : in("in"), out("out"), rc("rc"), lp("lp", 3.3e6), sp("sp"),
    rc_out("rc_out"), lp_out("lp_out")
    {
        rc.in(in);
        rc.out(rc_out);
        lp.in(rc_out);
        lp.out(lp_out);
        sp.in(lp_out);
        sp.out(out);
        }

    private:
        sca_tdf::sca_signal<double> rc_out, lp_out;
};
```

100 %



A hierarchy TDF example

Bit source

```
#include "systemc.h"
#include "systemc-ams.h"
#include <stdio.h>
using namespace std;

SCA_TDF_MODULE(bit_src)
{
    sca_tdf::sca_out<bool> out; // output port

    SCA_CTOR(bit_src) : out("out") {}

    void processing()
    {
        out.write( (bool)(std::rand()%2) );
    }
};
```

A hierarchy TDF example

Top level

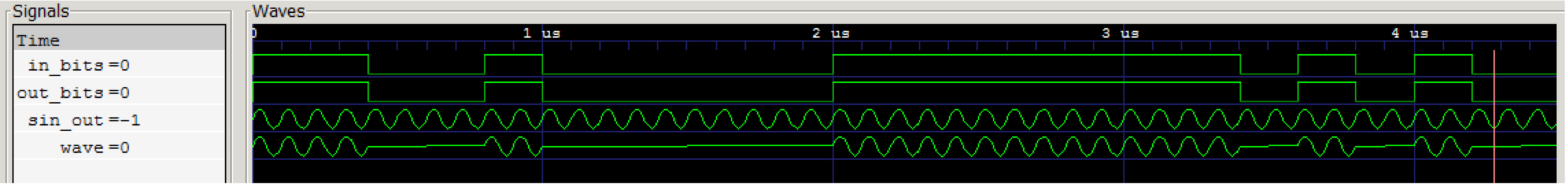
bit_src.h | rectifier.h | bask_demod.h | sine_wave.h | sampler.h | mixer.h | ltf_nd_filter.h | bask_mod.h | sim.cpp

BASK | (Global Scope) | sc_main(int argc, char * argv[])

```cpp
int sc_main(int argc, char* argv[])
{
    sc_core::sc_set_time_resolution(1.0, sc_core::SC_FS);

    sca_tdf::sca_signal<bool> in_bits, out_bits;
    sca_tdf::sca_signal<double> wave;
    sca_tdf::sca_signal<bool> wave_bool;
    sca_tdf::sca_signal<double> sin_out;

    bit_src bs("bs"); // random bit source
    bs.out(in_bits);

    bask_mod mod("mod"); // modulator
    mod.in(in_bits);
    mod.out(wave);
    mod.carrier(sin_out);

    bask_demod demod("demod"); // demodulator
    demod.in(wave);
    demod.out(out_bits);

    sca_util::sca_trace_file* atf = sca_util::sca_create_vcd_trace_file( "trace.vcd" );
    sca_util::sca_trace( atf, in_bits, "in_bits" );
    sca_util::sca_trace( atf, wave, "wave" );
    sca_util::sca_trace( atf, out_bits, "out_bits" );
    sca_util::sca_trace( atf, sin_out, "sin_out" );
    sc_core::sc_start(100, sc_core::SC_US);
    sca_util::sca_close_vcd_trace_file( atf );
        return 0;
}
```

100 %

A hierarchy TDF example



TDF execution semantics